Департамент образования Вологодской области
бюджетное профессиональное образовательное учреждение
Вологодской области
«ВОЛОГОДСКИЙ СТРОИТЕЛЬНЫЙ КОЛЛЕДЖ»

МЕТОДИЧЕСКИЕУКАЗАНИЯ

к практическим работам
по МДК 02.03. Проектирование приложений баз данных
ПМ.02. Участие в разработке информационных систем

09.02.04 Информационные системы (по отраслям)

2017

Рассмотрено на заседании предметно-цикловой комиссии общепрофессиональных, специальных дисциплин и дипломного проектирования по специальностям 08.02.01. Строительство и эксплуатация зданий и сооружений, 08.02.07. Монтаж и эксплуатация внутренних сантехнических устройств, кондиционирования воздуха и вентиляции, 43.02.08. Сервис домашнего и коммунального хозяйства

Данные методические указания предназначены для студентов специальности 09.02.04. Информационные системы (по отраслям) БПОУ ВО «Вологодский строительный колледж» при выполнении практических работ по МДК 02.03. Проектирование приложений баз данных

Объем практических работ по МДК 02.03.составляет **50** часов.

Оглавление

Практическая работа №13.Сервер баз данных Borland InterBase. SQL-сервер Local InterBase. Физическая организация базы данных формата InterBase. Организация сеанса связи с удаленной базой данных. Основы администрирования SQL-сервера Borland InterBase.

Практическая работа №14.Создание и перенос базы данных. Создание базы данных. Регистрация базы данных. Перенос базы данных из локальных БД в InterBase. Типы данных. Домены.

Практическая работа №15.Работа с таблицами. Создание, модификация и удаление таблиц. Изменение данных в таблицах.

Практическая работа №16.Работа с индексами. Создание, модификация и удаление индексов.

Практическая работа №17.Работа с представлениями. Создание, модификация и удаление представлений.

Практическая работа №18.Разработка клиентской части приложения. Размещение визуальных и не визуальных компонентов. Соединение с базой данных.

Практическая работа №19.Формирование SQL запросов для выборки данных. Простые и сложные запросы на выборку (сортировка, группировка, вычисляемые поля, составные операторы выборки). Создание SQL запросов для изменения наборов данных.

Практическая работа №20.Создание хранимых процедур. Команды по созданию, редактированию и удалению хранимой процедуры.

Практическая работа №21.Создание генератора и триггеров. Каскадные воздействия.

Практическая работа №22.Сортировка, поиск и фильтрация данных в базах данных и выборках.

Практическая работа №23.Обработка транзакций. Кэширование изменений. Работа с транзакциями в InterBase.

Практическая работа №24.Формирование и вывод отчетов. Назначение и виды отчетов. Компоненты для формирования отчетов.

Практическая работа №25.Установка привилегий доступа к данным. Программное администрирование баз данных InterBase.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

МДК 02.03. Проектирование приложений баз данных входит в состав профессионального модуля ПМ.02. Участие в разработке информационных систем в составе профессионального учебного цикла программы подготовки специалистов среднего звена по специальности 09.02.04 Информационные системы (по отраслям).

В результате освоения ПМ.02. обучающийся должен **иметь практический опыт:**

- использования инструментальных средств обработки информации;
- участия в разработке технического задания;
- формирования отчетной документации по результатам работ;
- использования стандартов при оформлении программной документации;
- программирования в соответствии с требованиями технического задания;
- использования критериев оценки качества и надежности функционирования информационных систем;
- применения методики тестирования разрабатываемых приложений;
- управления процессом разработки приложений с использованием инструментальных средств;

уметь:

- осуществлять математическую и информационную постановку задач по обработке информации, использовать алгоритмы обработки информа-ции для различных приложений;
- уметь решать прикладные вопросы интеллектуальных систем с использованием, статических экспертных систем, экспертных систем реального времени;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ, разрабатывать графический интерфейс приложения;
- создавать проект по разработке приложения и формулировать его зада-чи, выполнять управление проектом с использованием инструментальных средств;

знать:

- основные виды и процедуры обработки информации, модели и методы решения задач обработки информации (генерация отчетов, поддержка принятия решений, анализ данных, искусственный интеллект, обработка изображений);

- сервисно - ориентированные архитектуры, CRM-системы, ERP-системы;
- объектно-ориентированное программирование;
- спецификации языка, создание графического пользовательского интер-фейса (GUI), файловый ввод-вывод, создание сетевого сервера и сетевого клиента;
- платформы для создания, исполнения и управления информационной системой;
- основные процессы управления проектом разработки.

В соответствии с учебным планом на изучение МДК 02.03. Проектирование приложений баз данных отводится 105 часов, в том числе 50 часов – практические занятия.

Целью практических занятий является формирование практических умений, необходимых в последующей учебной и профессиональной деятельности.

Содержание практических занятий по МДК 02.03. Проектирование приложений баз данных направлено на реализацию требований Федерального государственного образовательного стандарта по специальности СПО 09.02.04 Информационные системы (по отраслям). Практическое занятие включает следующие структурные элементы:

1) инструктаж, проводимый преподавателем,
2) самостоятельная деятельность обучающихся,
3) анализ и оценка выполненных работ.

Контроль и оценка результатов выполнения обучающимися работ, заданий на практических занятиях направлены на проверку освоения умений, практического опыта, развития общих и формирование профессиональных компетенций, определённых программой учебной дисциплины.

Оценки за выполнение заданий на практических занятиях выставляются по пятибалльной системе и учитываются как показатели текущей успеваемости обучающихся.

ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

№ п/п	Тема программы	Тема работы	Количество часов
1.	*Тема 1. Разработка клиентских приложений баз данных*	Создание и редактирование псевдонимов баз данных. Создание баз данных. Создание новой таблицы, задание полей, задание свойств таблицы, изменение структуры и заполнение таблицы с помощью Database Desktop.	2
2.		Компоненты для работы с базами данных. Размещение и настройка основных компонентов, размещение и настройка панелей, настройка компонента DBGrid, формирование вычисляемого поля.	2
3.		Навигационный способ доступа к данным. Формирование основного меню. Методы для сортировки и поиска данных.	2
4.		Реляционный способ доступа к данным. Методы для фильтрации данных. Статические и динамические запросы.	2
5.		Формирование графика зависимости данных из БД. Основные методы и свойства DBChart. Настройка и печать графика.	2
6.		Работа с отчетами. Компоненты отчета, группирование данных в отчете. Создание отчета для связанных наборов данных.	2
7.		Разработка приложения для соединения данных двух таблиц 1:1. Методы объединения данных двух таблиц.	2
8.		Разработка приложения для формирования, слияния и разъединения однотипных таблиц баз данных. Основные методы и свойства компонента BatchMove.	2
9.		Разработка приложения с таблицей для выбора допустимых значений. Установка связи головной и вспомогательной таблиц при создании БД в Database Desktop. Поля просмотра lookup fields. Использование редактора полей при создании нового поля зависимой таблицы. Разработка приложения для таблиц, связанных с помощью свойства Referential Integrity.	2
10.		Разработка приложения для базы данных MS Access в Delphi. Основные свойства и методы компонента ADOConnection.	2
11.		Поиск, фильтрация и индексация таблиц. Последовательный перебор, метод Locate, метод Lookup. Фильтрация данных. Использование индексов.	2
12.		Основные свойства, события и методы набора данных. Компоненты TADOTable, TADOQuery или TADOStoredProc. Курсоры в наборах данных ADO.	

13.	*Тема 2. Разработка распределенных систем обработки информации*	Сервер баз данных Borland InterBase. SQL-сервер Local InterBase. Физическая организация базы данных формата InterBase. Организация сеанса связи с удаленной базой данных. Основы администрирования SQL-сервера Borland InterBase.	
14.		Создание и перенос базы данных. Создание базы данных. Регистрация базы данных. Перенос базы данных из локальных БД в InterBase. Типы данных. Домены.	2
15.		Работа с таблицами. Создание, модификация и удаление таблиц. Изменение данных в таблицах.	2
16.		Работа с индексами. Создание, модификация и удаление индексов.	2
17.		Работа с представлениями. Создание, модификация и удаление представлений.	2
18.		Разработка клиентской части приложения. Размещение визуальных и не визуальных компонентов. Соединение с базой данных.	2
19.		Формирование SQL запросов для выборки данных. Простые и сложные запросы на выборку (сортировка, группировка, вычисляемые поля, составные операторы выборки). Создание SQL запросов для изменения наборов данных.	2
20.		Создание хранимых процедур. Команды по созданию, редактированию и удалению хранимой процедуры.	2
21.		Создание генератора и триггеров. Каскадные воздействия.	2
22.		Сортировка, поиск и фильтрация данных в базах данных и выборках.	2
23.		Обработка транзакций. Кэширование изменений. Работа с транзакциями в InterBase.	2
24.		Формирование и вывод отчетов. Назначение и виды отчетов. Компоненты для формирования отчетов.	2
25.		Установка привилегий доступа к данным. Программное администрирование баз данных InterBase.	2

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКИХ РАБОТ

Практическая работа №1
Создание и редактирование псевдонимов баз данных

Цель работы: сформировать умения по созданию псевдонимов базы данных, файлов конфигурации; научится редактировать свойства созданных псевдонимов.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **Paradox**.

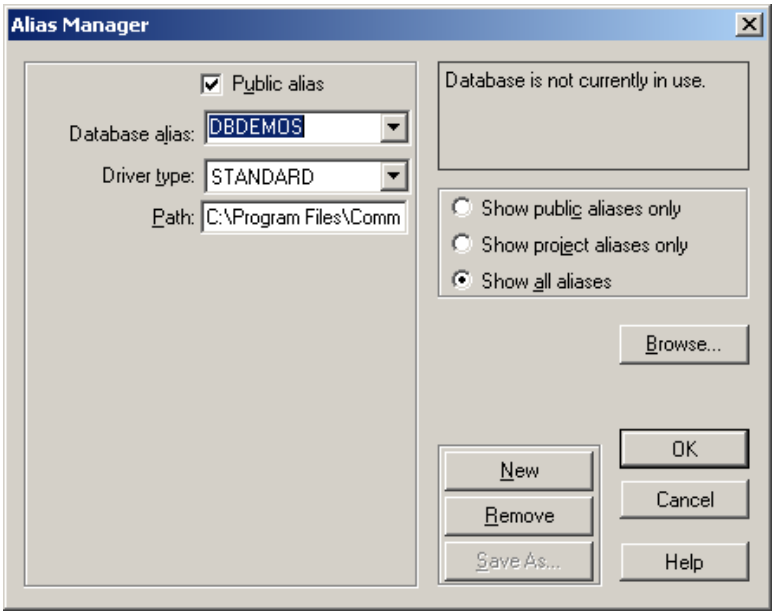
Задание 1.

Установить русификатор для работы с таблицами типа **Paradox**. Создать псевдоним новой базы данных и файл конфигурации.

Методические указания по выполнению задания:

1. В каталоге **IS-44** создайте подкаталог для размещения базы данных – **BASE.**

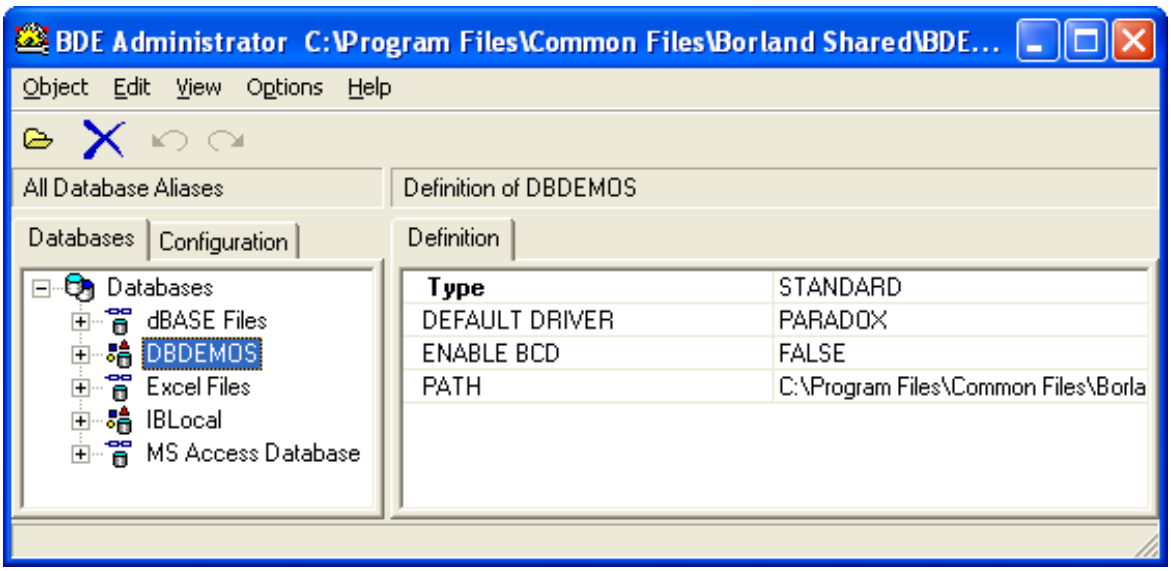
2. Базы данных идентифицируются логическими именами - псевдонимами. **Псевдоним** - это логическое имя базы данных. Он определяет каталог БД. Псевдонимы дают возможность перемещения таблиц данных в другое место на диске или внутри сети. Псевдоним (**alias**) определяет местонахождение файлов БД, т. е. имя каталога, в котором расположены файлы БД. Использование псевдонима существенно облегчает перенос файлов БД в другой каталог и приложения на другой компьютер.
3. Программа **Database Desktop** предназначена для создания и редактирования таблиц, SQL-запросов и для работы с псевдонимами. Эту программу можно вызывать из среды **Delphi** командой **Tools - Database Desktop**. Запустите программу **Database Desktop.**
4. Для работы с псевдонимами используется **менеджер псевдонимов**. Его надо вызвать командой **Tools - Alias Manager** из среды **Database Desktop**. Запустите менеджер псевдонимов. Вы увидите диалоговое окно диспетчера псевдонимов, вид которого представлен на рисунке. Вид этого окна существенно зависит от того, псевдоним какой базы данных просматривается.



5. С его помощью можно создавать или удалять псевдонимы. Могут создаваться псевдонимы двух типов: открытые, доступные при работе из любого каталога, и псевдонимы проекта, доступные только при работе в конкретном рабочем каталоге.
6. Имя псевдонима вводится (для нового) или выбирается (для существующего) из выпадающего списка окна **Database alias**. Кроме того, можно изменять параметры существующих псевдонимов: тип драйвера и путь к БД (поле окна **Path**). Путь можно

ввести вручную или выбрать в окне просмотра каталогов (после нажатия кнопки **Browse**).

7. Для создания псевдонима нажмите кнопки:
   - **New** - окно **Database Alias** с именем псевдонима очистится, поэтому можно ввести имея псевдонима – **STUD**, значение в окне **Driver Type** оставить равным **STANDART**;
   - **Browse** - появится форма **Directory Browser**; в ее окне **Drive (or Alias)** выбрать и открыть каталог с БД - **BASE** и нажать кнопку **ОК**; в окне **Path** формы **Alias Manager** появится путь к базе данных;
   - **Keep New** – сохранится псевдоним и путь к БД; в списке окна **Data base Alias** появится имя псевдонима БД;
   - **Save As** - появляется форма **Save Configuration File**; в ней будет предложено полное имя файла конфигурации: путь к нему и имя **IDAPI32.cfg**; в окне **Alias** нужно выбрать имя нового или требуемого псевдонима; в окне **Сохранить в** появится имя каталога с файлами БД; сохраним файл конфигурации в каталог **IS-44**; для этого надо:
     - в верхнем окне **Папка** перейти в каталог **IS-44**,
     - в окне **Тип файла** оставить тип файла **Config (*.cfg)**,
     - в окне **Alias** выбрать имя нового псевдонима и нажать кнопку **Сохранить**;
     - после перехода к форме **Alias Manager** нажать **ОК**.
8. Программа **BDE Administrator** позволяет настраивать параметры БД и операционной системы, в том числе параметры псевдонима (имя, тип, путь), драйвера (тип, язык) и системные установки (форматы даты, времени, числовые форматы). Запустите утилиту **BDE Administrator**, выполнив команду **Tools - BDE Administrator,** главного меню среды программирования **Turbo Delphi**.
9. Окно программы имеет две страницы: **Database** – базы данных и **Configuration** – конфигурация. На странице в левой панели расположено дерево псевдонимов баз данных.



10. Выделив интересующий вас псевдоним в левой панели, вы можете в правой панели **Definition** увидеть все его характеристики. Число и смысл этих характеристик зависит от используемого драйвера. Для драйвера **STANDART**, используемого, в частности, для баз данных **Paradox** набор характеристик минимален: **Type** – имя драйвера и **Path** – путь к базе данных. Щелкнув на параметре **Path**, вы увидите кнопку с многоточием, при её нажатии откроется стандартный диалог **Windows**, позволяющий выбрать новый каталог. Таким образом, вы можете изменить характеристики псевдонима, если, например, изменилось расположение базы данных на диске. После этого все приложения, использующие псевдоним, автоматически будут работать с данными, даже не заметив изменения их месторасположения. В правой панели можно изменять только те параметры, имена которых не выделены жирным шрифтом. Значения выделенных параметров изменять нельзя.
11. Для создания псевдонима БД выберите в левой части вкладку **Databases**. Затем выполните команду **Object - New**. Появится диалоговое окно **New Database Alias** для выбора типа драйвера.
12. Для локальных таблиц выбирают тип **Standard**. Нажмите кнопку **ОК**. Появляется окно для установки параметров псевдонима. На странице **Databases** в списке имен псевдонимов появляется имя нового псевдонима в виде **STANDARD1**. Его можно изменить на требуемое - **STUD1.**
13. На странице **Definition** появляется список параметров псевдонима: его тип (**STANDARD**); тип драйвера (формат таблиц - **PARADOX**); **ENABLE BCD** -

необходимость перевода чисел в формат **BCD** для повышения точности вычислений; **PATH** - путь к каталогу с БД.

14. Параметры псевдонима можно изменить. Измените имя псевдонима на **BASE1**. для этого щелкните по имени псевдонима правой кнопкой мыши и в контекстном меню выберите команду **Rename**.
15. Значение параметра **PATH** можно ввести вручную или выбрать с помощью окна **Select Directory** (выбор каталога). Это окно можно вызвать двойным щелчком в поле параметра **PATH** или щелчком на многоточии в конце строки со значением **PATH**. Многоточие появляется при выборе строки с параметром **PATH**. Укажите в качестве каталога базы данных каталог **BASE**, созданный ранее. После нажатия кнопки **ОК** путь автоматически заносится в качестве значения **PATH**. Измените каталог базы данных для псевдонима **BASE1** на **IS-44**.
16. Удалите псевдоним **BASE1** командой **Delete** контекстного меню или командой **Object - Delete** главного меню администратора.
17. Установите русификатор. Выберите в окне программы вкладку **Configuration**. Разверните список узлов и выберите узел **Drivers**.
18. Разверните список типов баз данных, выбрав узел **Native**. Выберите тип **PARADOX**. В правой части формы откроется страница определения параметров типа (**Definition**). Выберите строку с параметром **LANGDRIVER** (языковый драйвер). В правом столбце списка типов открыть список драйверов, выбрать драйвер **Pdox ANSI Cyrillic**. Примените внесенные изменения, выполнив команду **Object - Apply**.
19. Закройте окно **BDE Administrator**.

Задание 2.

Разработать структуру записи, которая включает следующие поля: NZ (номер зачетки), FIO (фамилия и инициалы), RS (размер стипендии), FOTO (фотография).

Методические указания по выполнению задания:

1. Для разработки структуры таблиц и их записей можно использовать программу **Database Desktop**. Она позволяет выполнять создание таблиц, изменение их структуры и редактирование ее записей. Действия по управлению структурой таблиц можно выполнить и программно.

2. Процесс создания новой таблицы начинается в среде **Database Desktop** командой **File – New - Table**. Появляется окно **Create Table** для выбора типа БД со значением **Paradox 7**, нажать **ОК**.
3. Далее появится форма **Structure Information Paradox 7 Table: (Untitled)**. В нем структура записи формируется в виде таблицы с заголовком **Field Roster: (список полей),** в каждой строке которой представлены сведения об одном элементе (поле, столбце) записи таблицы. Назначение столбцов таблицы:
   - **Номер элемента (поля) записи** - формируется автоматически;
   - **Field Name** - имя элемента записи - идентификатор;
   - **Type** - имя типа элемента записи; выбор типа элемента производить из выпадающего списка допустимых типов; вызвать список можно нажатием клавиши 'пробел'; основные типы записей: **Alpha** (строковый), **Number (Real)**, **$ (Money)** (денежный), **Short** (целый), **Long Integer** (длинный целый), **Graphic** (графический);
   - **Size** - для строковых данных - максимально допустимое количество символов;
   - **Key** - признак основного индекса ключа в виде звездочки; ее можно установить, например, нажатием клавиши **Пробел**. Основной ключ должен быть в первом элементе записи. Он должен быть уникальным. В нашем случае это будет поле **NZ.**
4. Введите название первого поля **NZ** в колонку **Field Name** и нажмите клавишу табуляции для перехода к следующей колонке. Нажмите пробел, чтобы утилита **DBD** показала список возможных типов, и выберите **Short**. Щелкните по полю **Key**. Нажмите клавишу пробел, чтобы создать по полю первичный ключ.
5. Продолжите ввод полей таблицы в соответствии с таблицей. Переход на следующую строку формируемой таблицы производится нажатием клавиши **Enter.**

FIO	Alfa (size = 22)
RS	Number
FOTO	Graphic

6. Для полей **NZ** и **FIO** установите флажок **Required Field**, которые означает, что при вводе очередной записи в эти поля обязательно должны быть помещены значения.
7. В таблице может быть любое количество вторичных индексов, по которым можно сортировать и искать данные для их показа в таблице. Для поля **FIO** определим индекс. Для этого раскройте список **Table properties** в правом верхнем углу окна, выберите пункт **Secondary Indexes** и щелкните на появившейся кнопке **Define.**

8. В окне **Define Secondary Index** из списка полей перенесите поле **FIO** в список **Indexed fields**. С помощью флажков группы **Index options** можно определить следующие особенности индекса: **Unique** – индекс будет содержать уникальные значения; **Maintained** – индексные поля сортируются по возрастанию значений; **Case sensitive** – индекс чувствителен к регистру букв в текстовых полях; **Descending** – индексные поля сортируются по убыванию значений. В нашем случае оставьте эти флажки без изменений и щелкните по кнопке **ОК**. Появится форма **Save Index As**. В поле **Index Name** введите имя вторичного индекса **Ind_FIO** и щелкните по кнопке **ОК**.
9. Выполните аналогичные действия для создания вторичного индекса **RS** (имя индекса – **Ind_RS**).
10. Выполните команду **Save as** для формирования имени таблицы с записями заданной структуры. В окне **Save Table as** выберите псевдоним **STUD** из списка в окне **Alias**, в поле **Save in** выберите и откройте каталог **IS-44\BASE** с таблицами БД.
11. После этого в окно **Имя файла** введите имя файла таблицы в базе данных - **Stud1**, и нажмите кнопку **Save.**
12. Перейдите в каталог **BASE**. Посмотрите, какие файлы, сколько и с каким расширением созданы.
13. Определить вторичный индекс для сортировки можно и при повторном входе в систему **Database Desktop**. Для этого выполнить команду вызова утилиты: **Tools – Utilites - Restructure**. Появится окно **Select File**. В его поле **Alias:** выбрать имя псевдонима базы данных. В поле **Папка:** появится имя открытого каталога с базой данных, а в основном окне - список имен ее таблиц. Выбрать и открыть требуемый файл. Появится окно **Restructure Paradox 7 Table: имя_таблицы** и список полей ее записей под заголовком **Field Roster**. Можно корректировать структуру записи таблицы, в том числе и вторичные индексы. Откройте таблицу **Stud1** и измените имя вторичного индекса **Ind_FIO** на **I_FIO**.
14. Закройте программу **Database Desktop.**

Внеаудиторная самостоятельная работа

Ответьте на вопросы письменно в тетради:

- Что такое псевдоним базы данных? Укажите, для чего используются псевдонимы.

- Опишите особенности работы в **DataBase Desktop** по созданию и изменению псевдонимов баз данных.
- Запишите алгоритм создания псевдонима базы данных в программе **BDE Administrator**.
- Каково назначение и основные возможности системы **Database Desktop?**
- Как разработать структуру записи таблицы базы данных с помощью системы **Database Desktop?**
- Каково назначение первичного и вторичного индексов таблицы базы данных?
- Как определить основной и вторичный индексы базы данных?

Практичсекая работа №2
Компоненты для работы с базами данных. Размещение и настройка основных компонентов, размещение и настройка панелей, настройка компонента DBGrid, формирование вычисляемого поля.

Цель работы: изучить особенности использования компонентов по работе с базами данных; сформировать умения по работе с компонентами для отображения элементов базы данных.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **Paradox**.

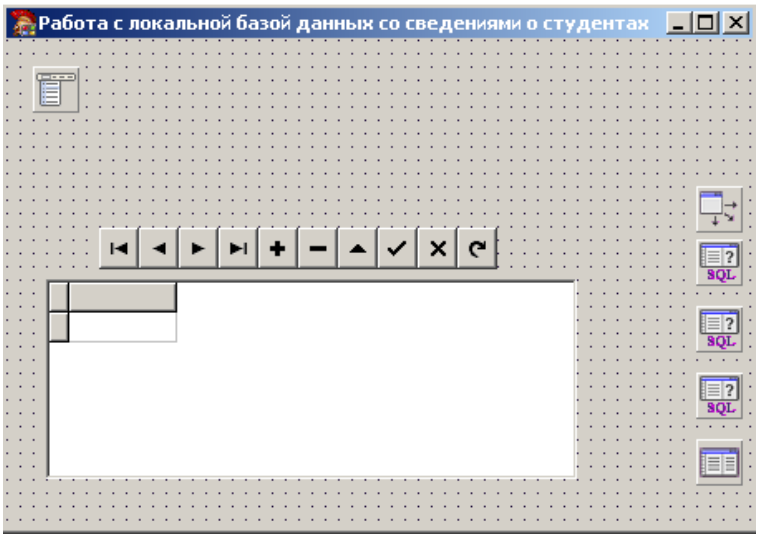
Задание 1. Размещение и настройка основных компонентов

Методические указания по выполнению задания:

1. Запустите интегрированную среду разработчика **Turbo Delphi**.
2. Сохраните проект в каталоге **Base**, под именем **BASE1.dpr**, выполнив команду **Save Project As**.
3. Установите свойство формы **Caption** равным **«Работа с локальной базой данных со сведениями о студентах».**
4. Пиктограммы компонентов для работы с БД расположены на страницах **BDE, Data Access, Data Controls, Decision Cube, QReport** и **InterBase**. На странице **BDE** размещены пиктограммы невизуальных компонентов доступа к данным, с помощью которых происходит связь приложения с физическими данными БД с использованием **BDE**. В том числе компоненты:
   - **Table** - компонент таблицы; обеспечивает доступ к таблице БД; он создает набор данных, который передает все данные из физической таблицы БД с помощью BDE;
   - **Query** - компонент запроса; предназначен для формирования набора данных из физической таблицы в соответствии с определенным запросом на языке SQL;
   - **StoredProc** - делает доступными процедуры, хранимые на сервере;
   - **Database** - устанавливает связь с БД;
   - **Session** - текущий сеанс работы с БД; для общего управления связью приложения с БД; Delphi автоматически генерирует объект Session в каждом приложении, работающем с БД;
   - **BatchMove** - для групповых операций переноса данных из одного набора в другой.
5. Набор данных (**DataSet**) - логическая таблица - это совокупность записей, взятых из одной или нескольких таблиц БД. Он может быть в одном из двух состояний - открытом или закрытом. Это определяет свойство **Active** типа **Boolean.**
6. На странице **Data Access** расположена пиктограмма невизуального компонента **DataSource**, предназначенного для связи компонентов типа **Table** и **Query** с

компонентами отображения данных. Остальные компоненты этой страницы служат для связи с **XML**-документами, кэширования и работы с пакетами.

7. Визуальные компоненты отображения данных **DBGrid, TDBNavigator, TDBText, TDBEdit, TDBMemo, TDBListBox, TDBComboBox, TDBCheckBox, TDBRadioGroup** подобны стандартным компонентам интерфейса пользователя, за исключением того, что их визуальное содержимое автоматически берется из соответствующих таблиц базы данных.
8. Разместить на основной форме компоненты: **TTable, TQuery (3 штуки), TDataSource, TDBGrid, TDBNavigator, TMainMenu.**
9. Выделите компонент **Table1**. Таблицы БД располагаются на диске и являются физическими объектами. Для операций с данными БД используются наборы данных (**DataSet**). Набор данных - это совокупность записей, полученных из одной или нескольких таблиц БД. Набор данных является логической таблицей, с которой работает приложение. Записи, входящие в набор данных, отбираются по определенным правилам. Компонент **Table** используется для навигационного доступа к данным.



10. Установите следующие значения свойств компонента **Table1**:
    - **DataBaseName** равняется **STUD** – задает имя БД, которое выбирается из выпадающего списка псевдонимов;
    - **TableName** равняется **Stud1.db** – задает имя физической таблицы БД; определяет местоположение данных БД; выбирается из выпадающего списка с именами всех основных таблиц выбранной БД;

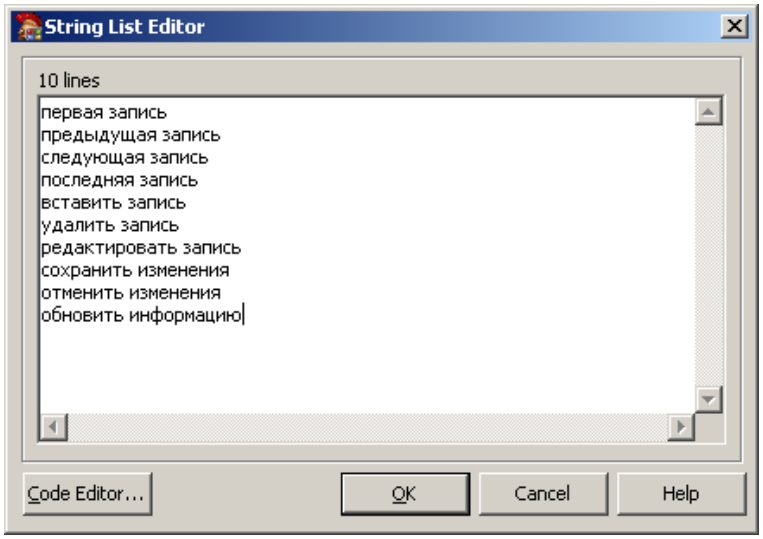
- **IndexFieldNames** равняется **NZ –** определяет имя поля для индекса - для сортировки по нему записей при их выводе в **DBGridl**;
- **Active** равняется **True** - активность таблицы; при **Active = True** набор данных находится в открытом состоянии; при **Active = False** - в закрытом. В процессе разработки можно установить свойство **Active = True** только после установки свойств **DatabaseName** и **TableName**, после чего система отображения данных заполняется значениями БД.

11. Выделите компоненты **Query1, Query2, Query3**. Компонент **Query** формирует набор данных с помощью запроса, определенного в его свойстве **SQL**. Компонент используется при реляционном (групповом) доступе к данным. Набор данных **Query** может включать записи более чем одной таблицы. **SQL**-запрос содержит команды (операторы) на языке **SQL**. Он выполняется при открытии набора данных. Запрос на языке **SQL** называют **SQL**-программой.
12. Установите следующие значения свойств компонентов **Query1, Query2, Query3:**
    - С помощью свойства **DatabaseName** компонент подключается к БД - **DataBaseName** равняется **STUD**;
    - Данные, отобранные в соответствии с запросом, определенным операторами свойства **SQL**, отображаются в процессе разработки приложения сразу после установки свойства **DataSet** компонента **DataSource**. Свойство компонент **DataSource** оставьте в виде пустой строки.
13. Выделите компонент **DataSource1**. Компонент **DataSource** является посредником при передаче данных от компонентов **Table** и **Query** к компонентам отображения (показа) данных. **DataSet** - основное свойство компонента, оно определяет, от какого из компонентов типа **TTable** и **TQuery** надо принять данные для показа. Значение этого свойства компонента можно установить при разработке приложения выбором имени компонента из выпадающего списка и при выполнении приложения. Установите свойство **DataSet** равным **Table1**.
14. Выделите компонент **DBNavigator** - компонент управления набором данных. Навигатор содержит кнопки для выполнения различных операций с набором данных путем автоматического вызова соответствующих методов. **DataSource** - основное свойство навигатора. Установите свойство **DataSource** равным **DataSourcel.**

15. Свойство **Flat** компонента **Navigator** управляет внешним видом навигатора. По умолчанию **Flat = False**, и кнопки отображаются в объемном виде. Установите **Flat = True**, какие изменения произошли с навигатором?
16. Состав видимых кнопок навигатора определяется свойством **VisibleButtons**. Это множественное свойство. Оно содержит для каждой из кнопок булевское свойство, определяющее их видимость. По умолчанию видимы все кнопки. Назначение кнопок приведено в таблице. Наименование кнопки содержит префикс **nb** и наименование метода, вызываемого этой кнопкой.

Пиктограмма	Наименование	Назначение кнопки
	**nbFirs**	Перейти к первой записи
	**nbPrior**	Перейти к предыдущей записи
	**nbNext**	Перейти к следующей записи
	**nbLast**	Перейти к последней записи
	**nbInsert**	Вставить новую запись
	**nbDelete**	Удалить текущую запись
	**nbEdit**	Редактировать текущую запись
	**nbPost**	Сохранить результат изменения записи
	**nbCancel**	Отменить изменения в текущей записи
	**nbRefresh**	Обновить информацию в наборе данных

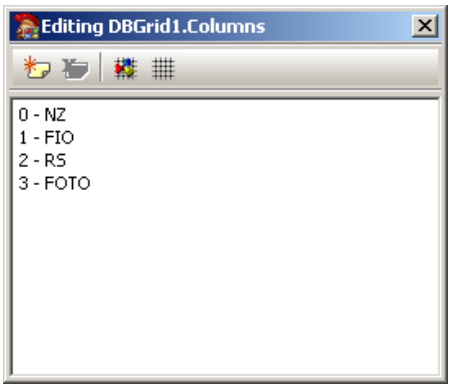
17. Подсказки для каждой кнопки навигатора содержит свойство **Hints**. Все подсказки записаны на английском языке. Их можно заменить текстами на русском языке. Для этого надо вызвать редактор **String List Editor** двойным щелчком в поле свойства **Hints** шли щелчком на многоточии в конце строки свойства. Установите тексты подсказок на русском языке.



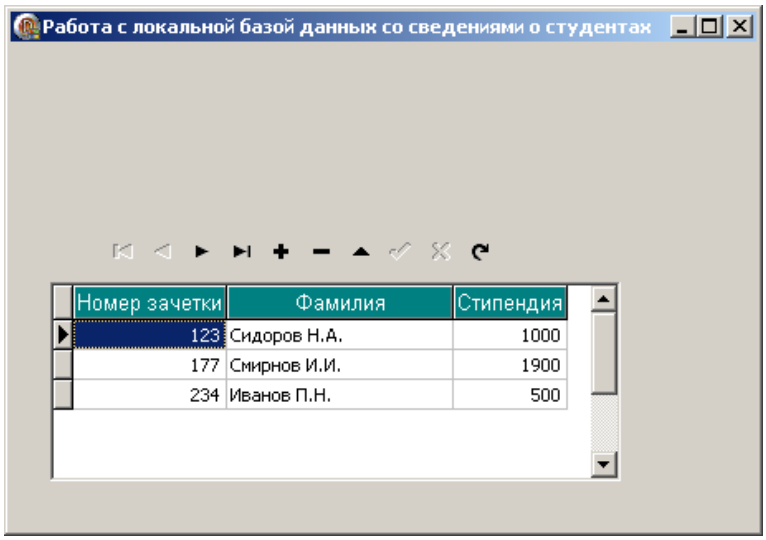
Задание 2. Настройка компонента DBGrid

Методические указания по выполнению задания:

1. Компонент **DBGrid** используют для вывода записей набора данных в табличном виде. Внешний вид сетки соответствует структуре записи таблицы БД. Строка сетки соответствует записи, столбец - полю. **DataSource** - основное свойство сетки. Оно устанавливается в процессе разработки приложения выбором из выпадающего списка источников и определяет **DataSet** - источник набора данных для отображения их на сетке. Его можно изменить в процессе выполнения приложения. Выделите компонент **DBGridl** и установите свойство **DataSource** равным **DataSourcel**.
2. Установить состав столбцов каждой строки таблицы компонента **DBGridl** можно с помощью ее дизайнера. Для вызова дизайнера **Editing DBGridl.Columns** надо на компоненте **DBGridl** щелкнуть дважды левой кнопкой или правой кнопкой вызвать контекстное меню, а в нем - пункт **Fields Editor**.
3. Вызвать контекстное меню дизайнера и выбрать команду **Add All Fields**. В окне редактора появятся все поля записи БД.



4. Выделите и удалите графическое поле **FOTO**. В окне **DBGridl** появятся все неудаленные поля таблицы.
5. Сформируйте дизайн компонента **DBGridl**: шрифт и цвет фона заголовков и данных его столбцов. Для этого в дизайнере **Editing DBGridl.Columns** выделите имя столбца **NZ.** На панели свойств данного поля выберите свойство **Title**: задайте **Caption** – **Номер зачетки, Aligment – taCenter, Color – clTeal.** Выберите свойство **Font:** задайте **Name – Arial, Size – 10, Color - clWindow**.
6. Выполните аналогичные действия для остальных полей, задав следующие имена **FIO – Фамилия, инициалы; RS – Стипендия**.
7. Запустите приложение на выполнение и проверьте возможность создания и корректировки текстовых элементов записей базы данных. Создайте несколько записей БД по образцу:

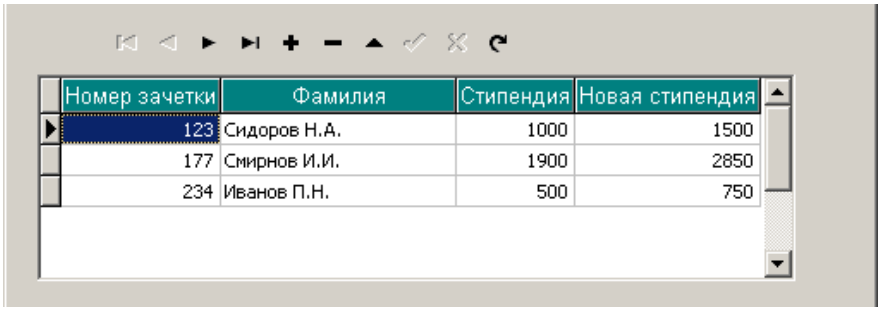


Задание 3. Создание вычисляемого поля

Методические указания по выполнению задания:

1. Создадим вычисляемое поле таблицы для расчета стипендии. Для этого нужно выбрать компонент **Table1**, щелкнуть на нем правой кнопкой мыши. Появится меню, выбрать в нем пункт **Fields Editor**. Появится форма **Forml.Table1** - редактор статических полей таблицы. На этом окне щелкнуть правой кнопкой мыши. Появится меню, выберите пункт **New field**. Появится окно для формирования нового поля. В этом окне надо определить **Name** - имя нового поля (**NRS**), **Type** - его тип - из списка (**Float**) и установить свойство **Calculated**. Нажать **ОК**. В окне **Forml.Table1** появится имя нового поля.

2. Выберите в окне **Forml.Table1** имя нового поля, в **Инспекторе объектов** появится список его свойств. Установить его свойства: **Display Label** - имя заголовка столбца нового поля в **DBGridl (NRS)**; **DisplayFormat** - формат вывода в окно **DBGrid, DisplayFormat = ###0.##** - 4 разряда до и 2 после запятой.
3. В окне **Forml.Table1** вызовите контекстное меню и выберите команду **Add all fields**. В окне **Forml.Table1** отобразятся все поля нашей таблицы.
4. Выберите компонент **Table1**. На странице **Events Инспектора объектов** выберите событие **OnCalcFields**. Вызовите заготовку метода для этого события и напишите команду для вычисления значения вычисляемого поля. Формальным параметром метода является свойство **DataSet**. Для обеспечения возможности использования метода любым из компонентов производится присоединение к этому свойству с помощью оператора **With DataSet do**. Затем размещается команда определения значения вычисляемого поля по значениям элементов записи с помощью оператора:
   FieldByName('NRS').AsFloat:=FieldByName('RS').AsFloat* 1.5;
5. Для включения нового столбца в таблицу **DBGridl** и настройки его параметров надо: открыть дизайнер **Editing.DBGrid.Columns**; добавить в него новый столбец командой **Add**; в **Инспекторе объектов** появятся его свойства; установить его свойство **FieldName** (**NRS**) и настроить цвета и шрифты нового столбца. Параметры данных и заголовка столбца с вычисляемым полем устанавливаются аналогично основным столбцам.
6. Запустите приложение на выполнение и проверьте его работоспособность.



Номер зачетки	Фамилия	Стипендия	Новая стипендия
123	Сидоров Н.А.	1000	1500
177	Смирнов И.И.	1900	2850
234	Иванов П.Н.	500	750

7. Для определения значений вычисляемого поля при работе с компонентами типа **TQuery** надо для каждого такого компонента сформировать вычисляемое поле и на странице **Events** инспектора объектов установить для события **OnCalcFields** имя разделяемого метода **TableCalcFields**.

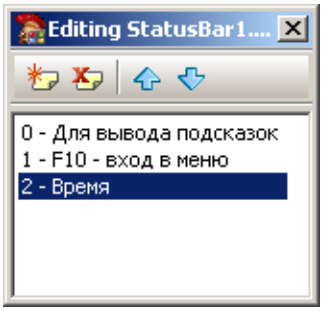
8. Выделите компонент **Query1**, в контекстном меню выберите команду **Fields Editor**, добавьте все существующие поля и создайте новое поле **NRS**.
9. Для события **OnCalcFields** компонента **Query1** запишите следующий программный код:

   Query1NRS.Value:=Query1RS.Value*1.5;
10. Сохраните изменения.

Задание 4. Размещение и настройка панелей

Методические указания по выполнению задания:

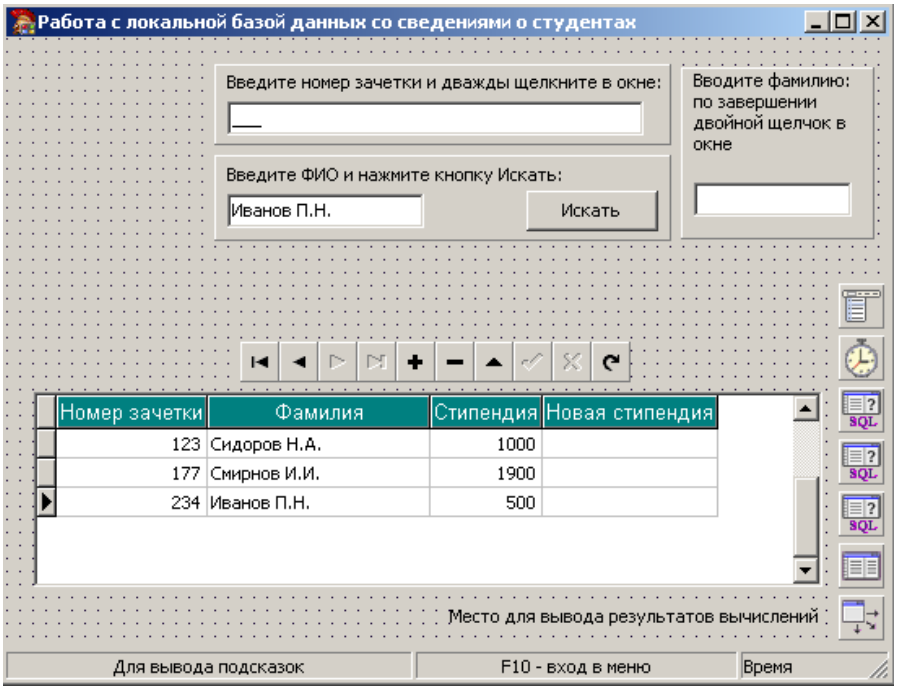
1. Разместить на форме компонент типа **TTimer** для показа текущего значения времени.
2. Разместите на форме метку типа **TLabell** для вывода сообщений. Установите ее свойства: **Visible = False, Caption = Место для вывода результатов вычислений.**
3. Разместите на форме компонент типа **TStatusBar**. Сформируйте на компоненте **StatusBarl** 3 панели. Для этого вызовите **Editing StatusBarl.Panels** - редактор панелей, выполнив двойной щелчок по соответствующему компоненту. Установите их свойства **Text** в соответствии с текстами, приведенными на рисунке.



4. Для вывода подсказок в нулевую панель компонента **StatusBarl** установите для компонента **StatusBarl** свойство **AutoHint = True**.
5. Для управления процессом отбора и поиска данных разместить на форме 3 панели.

- **Panel 1** - для размещения на ней компонентов для ввода фамилии для поиска. Разместите на ней:
  - метку с текстом: **Введите ФИО и нажмите кнопку Искать**;
  - текстовое поле **Editl** - для ввода требуемой фамилии. Установите свойство **Text** элемента управления равным **Иванов П. И.**;
  - кнопку типа **TButton**. Установите ее свойство **Caption** равным **Искать.**

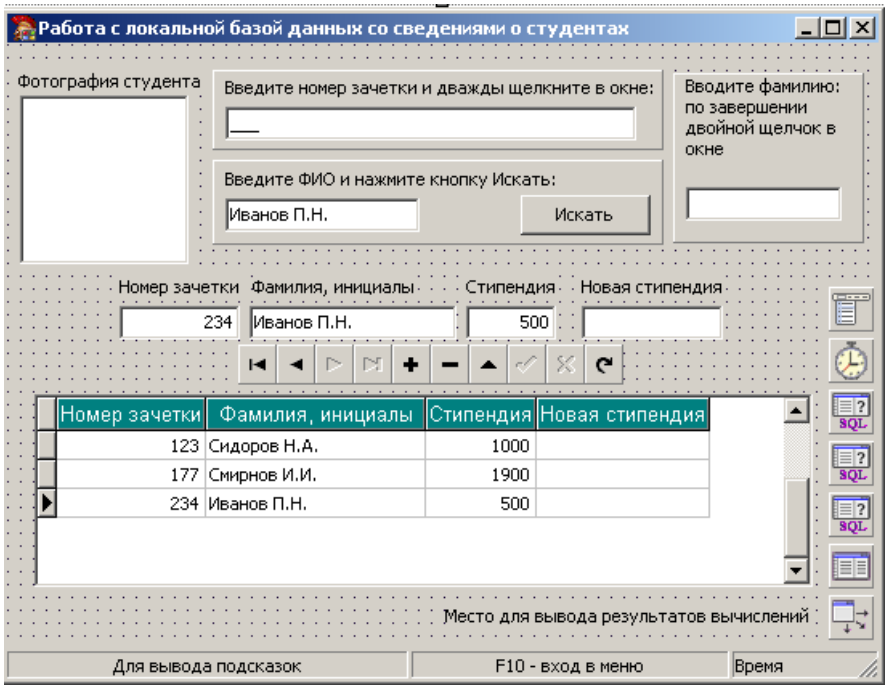
- **Panel 2** - для размещения на ней компонентов для ввода номера зачетки или размера стипендии для поиска. Разместите на ней:
  - метку с текстом **Введите номер зачетки и дважды щелкните в окне**;
  - **MaskEditl** - окно для ввода требуемого значения номера зачетки или размера стипендии; установите свойство **EditMask = 999;0** и свойство **Text = 170**;
- **Panel3** - для размещения на ней компонентов для ввода фамилии для постепенного поиска. Разместите на ней:
  - метку с текстом **Вводите фамилию: по завершении - двойной щелчок в окне:**;
  - поле **Edit2** - для ввода фамилии для постепенного поиска.



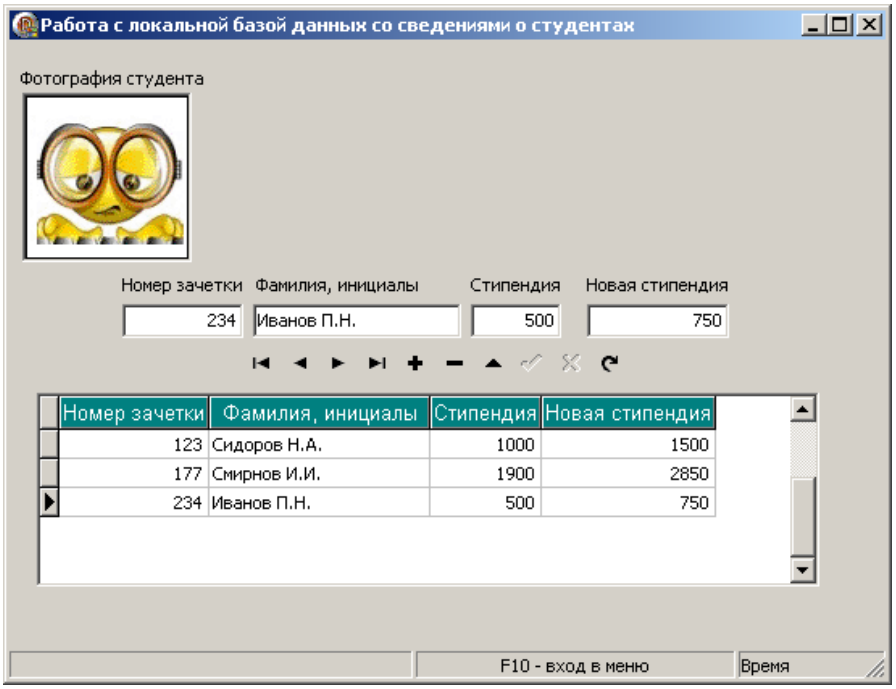
6. Установить свойства всех трех панелей **Visible = False**. Таким образом, при запуске приложения все панели скрыты. Они становятся видимыми только при выборе соответствующих пунктов меню.
7. Отображение данных может осуществляться с помощью окон, связанных с элементами записи БД. Для этого можно вручную для каждого элемента записи разместить на форме компоненты типа **TDBEdit, TDBImage** и **TLabel**. Но проще это сделать с помощью редактора полей **Fields Editor**. Для этого надо вызвать **Forml.Tablel** - редактор статических полей таблицы. Внести в него все поля, включая поле типа **TGraphic**. Командой **Select All** выделить все поля и «перетащить» их мышью на форму. На форме появятся окна для каждого элемента записи с метками над ними. Каждая метка

содержит имя поля записи. Разместить их должным образом и заменить их свойства **Caption** на осмысленные русские надписи. Пример формы с размещенными на ней окнами и надписями дан на рисунке.

8. **Stretch** - основное свойство компонента **DBImage** логического типа. Если значение свойства равно **True**, то размеры изображения подстраиваются под размеры компонента **DBImage** (искажая пропорции изображения), а если оно равно **False**, то изображение не изменяет свои размеры. Заполнение графического компонента производится в процессе выполнения приложения: из буфера или из файла с расширением **.bmp**. Для этого надо, например, заготовить в **ClipBoard** изображение. Действия при этом могут быть следующие: войти в какой-либо графический редактор (**Paint**); вызвать в него изображение из файла с расширением **.bmp**; скопировать из него в буфер **ClipBoard** фрагмент примерно такого же размера, что и окно компонента **DBImage1**; перейти в систему **Delphi**, запустить приложение на выполнение и вставить фрагмент из буфера **Clipboard** в компонент **DBImage1** выбранной записи командой **Shift+Ins** или **Ctrl+V**.



9. Вставьте изображения для каждой записи таблицы.



Внеаудиторная самостоятельная работа:

Составьте опорный конспект по основным командам.

Практическая работа №3
Навигационный способ доступа к данным. Формирование основного меню.
Методы для сортировки и поиска данных.

Цель работы: сформировать умения по созданию меню проекта; изучить особенности организации навигационного способа доступа к данным; изучить методы сортировки, фильтрации и поиска данных.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **Paradox**.

Теоретические сведения

Навигационный доступ состоит в том, что обрабатывается каждая запись набора данных. Перебор записей организуется программно. Достоинство этого способа доступа – простота. Недостаток – большая нагрузка на сеть, так как приложение получает все записи таблицы, независимо от того, сколько их требуется в приложении.

При работе с локальными БД предпочтительнее набор данных компонента **Table**. Он работает несколько быстрее, чем набор данных **Query**.

Каждый набор данных имеет указатель текущей записи, с полями которой могут быть выполнены операции редактирования и удаления. Методы компонентов **Table** и **Query** позволяют управлять положением этого указателя.

Навигационный способ доступа позволяет выполнение следующих операций: навигация по набору данных, сортировка записей, редактирование записей, вставка, удаление и фильтрация записей.

Задание 1. Формирование основного меню

Методические указания по выполнению задания:

1. Откройте проект **BASE1.dpr**.
2. Выполните двойной щелчок по компоненту **MainMenu**. Сформируйте основное меню в составе пунктов меню и подменю в соответствии с таблицей.

Пункт меню	Подменю 1	Подменю 2
Показать данные	все	
	отбора запросами	• с NZ>150 • с заданным NZ

		• с заданным запросом
	отбора фильтром	• для таблицы • для Query
Сортировать по:	• номерам зачеток • фамилиям • размеру стипендии	
Поиск	• по фамилии • быстрый по фамилии • ближайшей подходящей фамилии • постепенный поиск фамилии • по фамилии и стипендии	
Вычислить:	• сумму стипендий • среднюю стипендию	
График		
Отчет:	• просмотр • печать	
Выход		

3. Пункты меню дополните текстами свойства **Hint**, приведенными в таблице:

Наименование команды пункта меню	**Наименование пункта меню**	**Текст подсказки - свойство Hint**
Показать данные:	N1	Отбор данных разными способами
• все	N8	Показать все данные базы данных
• отбора запросами	N9	Отбор с помощью свойства SQL Query
• с NZ> 150	N11	Отбор со статическим запросом Query 1
• с заданным NZ	N12	Отбор динамическим запросом Query2
• с заданным запросом	N15	Отбор формируемым запросом Query3
• отбора фильтром	N10	Обор данных фильтрами
• для таблицы	N13	Отбор фильтром, сформированным для Tablel.Filter
• для Query	N14	Отбор фильтром, сформированным для Queryl.Filter
Сортировать по:	N2	Сортировка по разным столбцам
• номерам зачеток	N16	Сортировать данные по номерам зачеток
• фамилиям	N17	Сортировать данные по фамилиям
• размеру стипендии	N18	Сортировать данные по размеру стипендии
Поиск	N3	Поиск данных разными способами
• по фамилии	N19	Найти строку по заданной фамилии
• быстрый по фамилии	N20	Найти строку по заданной фамилии быстрым способом
• ближайшей подходящей фамилии	N21	Найти строку с фамилией, ближайшей подходящей заданной
• постепенный поиск	N22	Искать строку по мере ввода фамилии в окно

фамилии		
• по фамилии и стипендии	N23	Найти строку по заданным фамилии и стипендии
Вычислить:	N4	Работа с непосредственным доступом к базе данных
• сумму стипендий	N24	Вычислить сумму данных столбца
• среднюю стипендию	N25	Вычислить среднюю стипендию
График	N5	Показать график зависимости стипендий от номеров зачеток
Отчет:	N6	Формирование отчета с данными базы
• просмотр	N26	Показать отчет для предварительного просмотра
• печать	N27	Отпечатать отчет
Выход	N7	Завершить выполнение приложения

Задание 2. Методы для сортировки данных

Методические указания по выполнению задания:

1. При вызове команд подменю **Сортировать по:** выполняется сортировка после установки соответствующего значения имени поля индекса **Table1.IndexFieldNames := 'NZ';** - для сортировки по номеру зачетки; **Table1.IndexFieldNames:='FIO';** - для сортировки по фамилиям**; Tablel.IndexFieldNames := 'RS';** - для сортировки по размеру стипендии.
2. Для определения поля индекса сортировки можно использовать разные методы для каждого пункта меню. Можно использовать и разделенный - один метод для всех пунктов меню, вызывающих сортировку. В нем определение поля индекса сортировки производится методом **N16Click**. Выбор имени поля таблицы для формирования значения **IndexFieldNames** - имени поля индекса - производится из списка значений свойства **IndexDefs**.

   Tablel.lndexFieldNames := Table1.IndexDefs[Tag].Fields;
3. Номер элемента списка определяется значением свойства **Tag**, установленного в процессе разработки меню для каждой команды подменю **Сортировать по:** Значения свойства **Tag** для сортировки по различным столбцам даны в следующей таблице.

Для сортировки по:	Имя пункта меню	Значение свойства Tag
номерам зачеток	N16	0
фамилиям	N17	2
размеру стипендии	N18	1

4. Последовательность индексов в списке **IndexDefs** можно видеть при вызове значений свойства **IndexDefs** компонента **Tablel.** Щелчок на многоточии этого свойства вызывает

Editing Tablel.IndexDefs - редактора индексов таблицы. Его вид представлен на рисунке, в котором видны номера элементов списка **IndexDefs**.

5. Откройте редактор меню **Form1.MainMenu1** и выполните двойной щелчок по пункту **Сортировать по: - номерам зачеток,** откроется окно с заготовкой процедуры. Вставьте в заготовку следующий программный код:

```
DataSource1.DataSet := Table1;
IF Table1.Filtered = True then Table1.Filtered := False;
With Sender as TMenuItem do
Table1.IndexFieldNames := Table1.IndexDefs[Tag].Fields;
```

6. Откройте редактор меню **Form1.MainMenu1** и выполните двойной щелчок по пункту **Сортировать по: - фамилиям,** откроется окно с заготовкой процедуры. Вставьте в заготовку следующий программный код:

```
DataSource1.DataSet := Table1;
IF Table1.Filtered = True then Table1.Filtered := False;
Table1.IndexFieldNames :='FIO';
```

7. Откройте редактор меню **Form1.MainMenu1** и выполните двойной щелчок по пункту **Сортировать по: - размеру стипендии,** откроется окно с заготовкой процедуры. Вставьте в заготовку следующий программный код:

```
DataSource1.DataSet := Table1;
IF Table1.Filtered = True then Table1.Filtered := False;
Table1.IndexFieldNames :='RS';
```

8. Сохраните проект. Поверьте работоспособность приложения.

Задание 3. Методы для поиска данных

Методические указания по выполнению задания:

1. Поиск данных производится по одной из команд пункта меню **Поиск**: **по фамилии (J=0); быстрый по фамилии (J=1); ближайшей подходящей (J=2); по фамилии и стипендии (J=3); постепенный поиск**. Для определения метода поиска используется значение свойства **Tag** пункта меню, выбранного пользователем для поиска. Задайте значение данного свойства для каждого пункта меню.
2. В окне модуля перейдите в раздел описания переменных **var** и опишите переменную **J: integer;**
3. Процесс поиска первых четырех видов реализуется в 2 этапа. На первом этапе с помощью метода **N19Click** (одного для всех методов поиска, кроме постепенного) производится: показ панели **Panel2** с меткой **Label2** и окном редактора **Edit1** для ввода

требуемой фамилии; установка **J**-значения признака метода поиска. Для поиска по фамилии и стипендии, кроме панели **Panel2**, устанавливается видимой также панель **Pane13**. В ее метку выводится текст **Введите размер стипендии:** и активизируется окно **MaskEdit2** для ввода требуемого размера стипендии.

4. Для этого откройте редактор меню **Form1.MainMenu1** и выполните двойной щелчок по пункту **Поиск – по фамилии,** откроется окно с заготовкой процедуры. Вставьте в заготовку следующий программный код:

```
with Sender as TMenuItem do begin
J:=Tag;
Panel1.Visible := True;
Label2.Visible := True;
Edit1.Visible := True;
Poisk.Visible :=True;
If J=3 then
   begin
   Panel3.Visible := True;
   Edit2.Visible := True;
   Label4.Visible := True;
   Label4.Caption := 'Введите размер стипендии:'
   end;
end;
```

5. Аналогичный код вставьте для пунктов меню **Поиск – быстрый по фамилии, Поиск - ближайшей подходящей фамилии, Поиск – по фамилии и стипендии**.
6. На втором этапе методом **Button1.Click** производится поиск записи одним из заданных методов поиска. В методе **N19Click** устанавливается значение свойства **J = Tag** соответствующего пункта меню. Затем производится показ панели **Panel1** с меткой **Label2** (**Введите ФИО и нажмите кнопку Искать**), окном **Editl** и кнопкой **Искать** (поменяйте свойство **Name** кнопки на **Poisk**) для запуска процесса поиска.
7. После ввода требуемой фамилии и инициалов и нажатия кнопки **Искать** начинается процесс поиска методом **Poisk.Click**. Он выполняется по-разному, в зависимости от значения признака **J**. После поиска данных выдается сообщение об успешности (или неуспешности) поиска с помощью вложенных процедур **S1** и **S2**.
8. По завершении поиска устанавливается невидимой панель **Panel2** оператором: **Panel2.Visible := False;**
9. Для **J=0** выполняется поиск по фамилии с помощью методов **Table1.SetKey** и **Table1.GoToKey**. Предварительно устанавливается значение поля индекса, по которому производится поиск, оператором: **Table1.IndexFieldNames:= 'FIO';** Значение

поискового признака устанавливается оператором: **Table1.FieldByName('FIO'). AsString :=Editl.Text;**

10. Для этого выполните двойной щелчок по кнопке **Искать** и в заготовке процедуры введите следующий программный код:

```
procedure TForm1.poiskClick(Sender: TObject);
 procedure S1;
   begin
     MessageDlg ('Запись найдена', mtInformation, [mbOK], 0);
   end;
 procedure S2;
   begin
     Beep;
     MessageDlg ('Запись не найдена', mtInformation, [mbOK], 0);
   end;
begin
 case J of
 0: begin
 Table1.IndexFieldNames :='FIO';
 Table1.SetKey;
 Table1.FieldByName('FIO').AsString := Edit1.Text;
 IF Table1.GotoKey then S1 else S2;
 end;
end;
end;
```

11. Для быстрого поиска (**J=1**) используется функция: **Table1.Locate('FIO',Editl.Text, [LoCaseInsensitive, LoPartialKey]);** где: **FIO** - имя поля записи, по которому выполняется поиск; **Editl.Text** - требуемое значение поля (фамилии); **LoCaseInsensitive** - регистр букв не учитывается; **LoPartialKey** - допускается частичное совпадение значений.
12. Дополните процедуру поиска следующими строками:

```
1: IF Table1.Locate('FIO', Edit1.Text,[LoCaseInsensitive, LoPartialKey])
then S1 else S2;
```

13. При поиске ближайшего наиболее подходящего значения последовательность операторов аналогична последовательности операторов при поиске по фамилии, за исключением: последним оператором является: **Tablel.GoToNearest**; - найти ближайшее подходящее значение.
14. Дополните процедуру поиска следующими строками:

```
2: begin
 Table1.IndexFieldNames := 'FIO';
 Table1.SetKey;
 Table1.FieldByName('FIO').AsString := Edit1.Text;
 Table1.GotoNearest;
 end;
```

15. При поиске по фамилии и стипендии используется метод-функция аналогично быстрому поиску; но в качестве фактического параметра с поисковыми признаками применен параметр в виде списка из двух значений для поиска: **Table1.Locate('FIO;RS', VarArrayOf([Editl.Text, MaskEditl.Text]), [LoCaseInsensitive, LoPartialKey])**
16. Дополните процедуру поиска следующими строками:

```
3:
 begin
 if Table1.Locate
('FIO;RS',VarArrayOf([Edit1.Text, MaskEdit1.Text]),[LoCaseInsensitive, LoPartialKey])
 then S1 else S2;
 Panel1.Visible := False;
 end;
```

17. Постепенный поиск производится в 3 этапа. Подготовка к поиску производится с помощью метода **N22Click - Поиск - постепенный**. С его помощью становится видимой панель **Panel3** с меткой (**Вводите фамилию: по завершении - двойной щелчок в окне:**) и с окном редактора **Edit2** - для ввода фамилии:

```
Panel3.Visible := True;
Edit2.Visible :=True;
Label4.Visible :=True;
```

18. Далее выделите элемент **Edit2** и в окне **Events** выберите событие **OnChange**. В заготовке процедуры запишите представленные ниже операторы: **Table1.IndexFieldNames := 'FIO'; Table1.FindNearest([Edit2.Text]);**
19. По завершении постепенного поиска фамилии и двойного щелчка в окне **Edit2** вызывается метод **Edit2.DblClick**, с помощью которого панель **Panel3** делается невидимой оператором: **Panel3.Visible := False;**
20. Далее добавим программный код для отображения текущего времени в строке состояния нашего приложения, для этого выполните двойной щелчок по иконке элемента управления **Timer**. В заготовке процедуры запишем следующие операторы: **StatusBar1.Panels[2].Text := TimeToStr (Time);**
21. Сохраните проект. Проверьте работоспособность нашего приложения.

Внеаудиторная самостоятельная работа:

Ответьте на вопросы письменно в тетради:

- Каково назначение первичного и вторичного индексов таблицы базы данных?
- Как определить первичный и вторичный индексы таблицы базы данных?
- Как задать сортировку по одному из индексов?

Практическая работа №4

Реляционный способ доступа к данным. Методы для фильтрации данных. Статические и динамические запросы

Цель работы: сформировать умения по созданию команд меню проекта; изучить особенности организации реляционного способа доступа к данным; изучить методы сортировки, фильтрации и поиска данных.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **Paradox**.

Теоретические сведения

Реляционный способ доступа к данным основан на операциях с группой записей. Для определения этих операций используется SQL – язык структурированных запросов. Поэтому реляционный доступ называют SQL-ориентированным. Для его реализации в Delphi применяются компоненты **Query** и **StoredProc**, позволяющие выполнить SQL-запрос.

Средства SQL применимы для выполнения операций с локальными и удаленными БД. Основное достоинство реляционного способа доступа – это небольшая загрузка сети, так как по сети передаются только запросы и результат их выполнения. Наиболее эффективны они при работе с удаленными БД. В локальных БД с помощью SQL-запроса можно:

- формировать составы полей набора данных при выполнении приложения;
- включать в набор данных поля и записи одной или нескольких таблиц;
- отбирать записи по сложным критериям;
- сортировать набор данных по любому полю, в том числе и неиндексированному;
- осуществлять поиск данных по полному или частичному совпадению критерия поиска со значениями поля записи.

Задание 1. Методы для отбора данных

Методические указания по выполнению задания:

4. Откройте проект **BASE1.dpr**.
5. Отбор данных для показа производится с помощью фильтров компонентов **Tablel** и **Query1** и с помощью статического и динамических запросов компонентов **Query2** и **Query3**. Показ всех данных производится с помощью метода **N8Click**. Для этого устанавливается запрет фильтрации данных с помощью компонента **Table1** оператором: **Tablel.Filtered := False;** - в случае, если она была разрешена. Затем формируется подключение данных к компонентам просмотра с помощью **Tablel** оператором: **DataSourcel.DataSet := Tablel;**
6. Откройте редактор меню **Form1.MainMenu1** и выполните двойной щелчок по пункту **Показать данные – все,** откроется окно с заготовкой процедуры. Вставьте в заготовку следующий программный код:

```
IF Table1.Filtered = True then Table1.Filtered := False;
DataSource1.DataSet := Table1;
```

7. Отбор данных по запросам производится с помощью компонентов **Query1**, **Query2** и **Query3**. В их свойстве SQL устанавливаются критерии (условия) отбора данных. Отбор данных с заранее заданным условием отбора данных производится методом **N11Click** с помощью статического запроса компонента **Query1**. Для этого выделите компонент **Query1** и в свойстве **SQL** компонента запишите текст запроса: **SELECT * FROM Stud1 WHERE Stud1.NZ > 150,** где **Studl -** имя файла таблицы БД. Затем установите свойство **Active** компонента **Query1** равным **True**.
8. При выборе команды пункта меню **Показать данные - отбора запросами – с NZ>150** для отбора данных активизируется компонент **Query1** и производится подключение данных к компонентам их просмотра с помощью **Query1.**
9. Откройте редактор меню **Form1.MainMenu1** и выполните двойной щелчок по пункту **Показать данные - отбора запросами – с NZ>150,** откроется окно с заготовкой процедуры. Вставьте в заготовку следующий программный код:

```
IF Table1.Filtered = True then Table1.Filtered := False;
If Query1.Active = False THEN Query1.Active := True;
DataSource1.DataSet := Query1;
```

10. В процессе выполнения приложения производится также отбор данных по заданному значению номера зачетки: по динамическому запросу (с помощью компонента **Query2**, пункт **N12**); с помощью динамически формируемого фильтра компонента **Table1 (N13);** с помощью динамически формируемого фильтра компонента **Query1 (N14)**; с помощью динамически формируемого запроса компонента **Query 3 (N15)**. Отбор данных для заданного значения **NZ** производится в 2 этапа: подготовка к отбору и выполнение операторов для реализации процесса отбора данных. Подготовка к отбору производится в методе **N12Click**. Это - один разделенный метод для всех перечисленных способов отбора. Для пунктов меню **N13, N14, N15** он выбирается из списка методов для **OnClick** - события пунктов меню на странице **Events**. В методе **N12Click** производится показ панели **Panel2** с меткой **Введите номер зачетки:** и компонентом **MaskEditl**. При этом используется одно и то же окно для ввода требуемого значения номера зачетки или размера стипендии. В этом методе устанавливается признак пункта меню, которым вызван метод. Это производится по значению свойства Tag пункта меню, которое определяет значение глобальной переменной **J** - признака способа отбора. Таким способом идентифицируется пункт меню. Значения свойства **Tag** соответствующих пунктов меню устанавливаются в процессе разработки приложения. Соответствие

пунктов подменю **Показать данные** и их значений свойства **Tag** дано в следующей таблице.

Наименование и назначение пункта меню	**Tag**
Отбор запросами - с заданным NZ (динамический запрос с Query2)	0
Отбор запросами - с заданным запросом (SOL для Query3)	1
Отбор фильтром — для таблицы (Filter для Table1)	2
Отбор фильтром - для Query (Filter для Query1)	3

11. Откройте редактор меню **Form1.MainMenu1** и выполните двойной щелчок по пункту **Показать данные – отбора запросами – с заданным NZ,** откроется окно с заготовкой процедуры. Вставьте в заготовку следующий программный код:

```
With Sender as TMenuItem do
begin
J := Tag;
Panel2.Visible := True;
Label3.WordWrap := True; Label3.Height := 39; Label3.Width := 146;
Label3.Caption := 'Введите номер зачетки и дважды щелкните в окне';
end;
```

12. Аналогичный код вставьте для пунктов меню **Показать данные – отбора запросами – с заданным запросом, Показать данные – отбора фильтром – для таблицы, Показать данные – отбора фильтром – для Query**.
13. После ввода требуемого значения номера зачетки и двойного щелчка в окне **MaskEditl** вызывается метод **MaskEditlDblClick**, который в зависимости от значения **J** выполняет отбор данных. Отбор данных по условию, которое формируется во время выполнения приложения, производится с помощью компонента **Query2** с динамическим запросом (**J=0**). Для формирования динамического запроса в свойство **SQL** компонента **Query2** введите текст оператора отбора данных:

    SELECT * FROM Stud1 WHERE Stud1.NZ = :TNZ, где **TNZ** - это формальный параметр, значение для которого надо ввести во время выполнения приложения; пробел перед **TNZ** недопустим.
14. После ввода текста свойства **SQL** надо вызвать **Editing Query2.Params** - редактор его параметра - нажатием на многоточие свойства **Params**. Затем активизировать строку с именем параметра (**TNZ**); в **Инспекторе объектов** появятся его свойства. Установить его свойство **Data Type** выбором из списка его значений **ftInteger**.

15. Далее выделите элемент **MaskEdit1**. Перейдите на вкладку **Evens,** найдите событие **OnlDblClick** и откройте окно редактора кода.
16. Запишите в появившейся заготовке следующий программный код:

```
procedure TForm1.MaskEdit1DblClick(Sender: TObject);
var A: integer;
begin
CASE J of
0: begin
  If Query2.Filtered = True then Query2.Filtered := False;
  DataSource1.DataSet := Query2;
  A := StrToInt (MaskEdit1.Text);
  With Query2 do begin
Close;
//Query2.Active := False;
//ParamByName ( 'TNZ' ). Value := A;
Params [0].AsInteger :=A;
Open;
//Query2.Active := True;
End;
end;
end;
end;
```

17. В методе **MaskEditlDblClic**k для пункта меню **отбора запросами - с заданным запросом** (для **J = 1**) производится стирание текста прежнего запроса методом **Query3.SQL.Clear**;
18. Затем формируется новый запрос с использованием значения номера зачетки из окна **MaskEditl** оператором:

    Query3.SQL.Add ('SELECT * FROM Stud1 WHERE Studl.NZ >' + MaskEditl.Text);
19. Далее выделите элемент **MaskEdit1**. Перейдите на вкладку **Evens,** найдите событие **OnlDblClick** и откройте окно редактора кода.
20. Отредактируйте, появившуюся заготовку вставив следующий фрагмент:

```
1: begin
  DataSource1.DataSet := Query3;
  Query3.Close;
  Query3.SQL.Clear;
  Query3.SQL.Add
  ('SELECT * FROM Stud1 WHERE Studl.NZ >' + MaskEdit1.Text);
  Query3.Open;
end;
```

21. Для отбора данных с помощью фильтра компонента **Table1** в окно **MaskEditl** вводится требуемое значение номера зачетки. По двойному щелчку в окне **MaskEditl** вызывается

метод **MaskEditlDblClick**, в котором по значению **J = 3** устанавливается фильтр для работы с компонентом **Table1** операторами:

DataSourcel.DataSet := Table1;

Table1.Filtered := True;

Tablel.Filter := 'NZ >' + MaskEditl.Text;

22. Аналогично устанавливается фильтрация данных с помощью компонента **Query1**. Дополнительно перед установкой фильтра **Query1** производится отключение **Query1** от БД оператором **Query1.Close;** а после установки фильтра - подключение **Query1** к БД оператором **Queryl.Open;**
23. Выделите элемент управления **MaskEdit1**. Перейдите на вкладку **Evens,** найдите событие **OnlDblClick** и откройте окно редактора кода.
24. Отредактируйте код, вставив следующий фрагмент:

```
2: begin
  DataSource1.DataSet := Table1;
  If Table1.Filtered = True then Table1.Filtered := False;
  Table1.Filter := 'NZ >' + MaskEdit1.Text;
end;
3: begin
  DataSource1.DataSet := Query1;
  Query1.Close;
  Query1.Filter := 'NZ >' + MaskEdit1.Text;
  If Query1.Filtered = True then Query1.Filtered := False;
  Query1.Open;
end;
end;
Panel2.Visible :=False;
end;
```

25. Сохраните изменения в проекте и заройте приложение.

Внеаудиторная самостоятельная работа:

Создайте приложения для обработки данных ведомости отгрузки товара со склада.

Наименование товара	Наличие товара до отгрузки	Количество отгруженного товара	Цена за единицу товара	Стоимость отгруженного товара	Остаток товара на складе

Практическая работа №5

Формирование графика зависимости данных из БД. Основные методы и свойства DBChart. Настройка и печать графика.

Цель работы: изучить основные методы и свойства компонента **DBChart**; сформировать умения по настройке компонента **DBChart** и построению графиков зависимостей данных базы данных.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

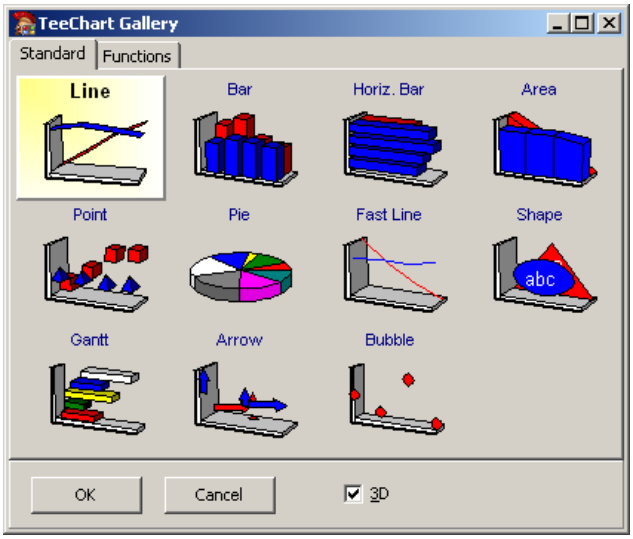
Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **Paradox**.

Задание 1. Методы команды меню «График» и настройка графика.

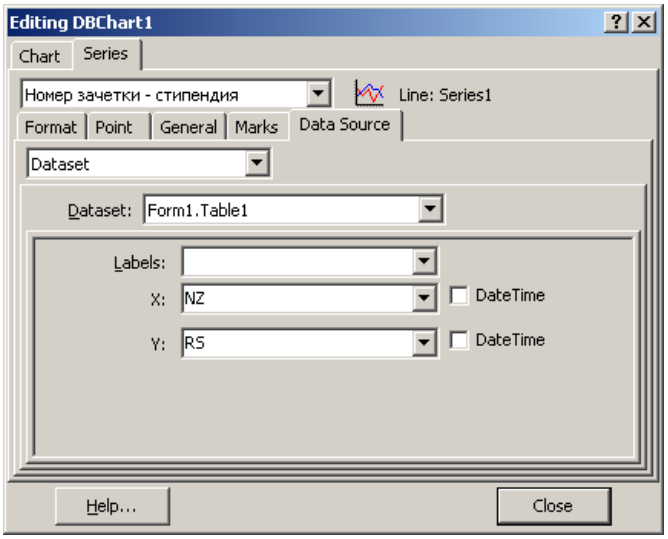
Методические указания по выполнению задания:

26. Откройте проект **BASE1.dpr**.
27. В рассматриваемом нами примере график выводится на отдельной форме **Form2**. Добавьте в проект новую форму, для этого выполните команду **File – New – Form**.
28. Установите свойство **Caption** данной формы равным **Вывод графиков по данным базы данных**.

29. Перейдите в главной форме проекта. К модулю главной формы проекта подключите модуль формы с графиком **Form2**, выполнив команду **File – Uses Unit**.
30. Для того чтобы форма с графиками открывалась в процессе выполнения программы необходимо в методе для пункта меню **График** записать следующий оператор **Form2.Show**.
31. Перейдите к форме **Form2**. К модулю данной формы подключите модуль главной формы, выполнив команду **File – Uses Unit**.
32. На форму установите пустую панель со свойством **Align**, равным **alTop**.
33. Для построения графиков и диаграмм используются компоненты **Chart** с вкладки **Additional**, и **DBChart** с вкладки **Data Controls** палитры компонентов. Это равноценные компоненты, отличие состоит в том, что **DBChart** принимает данные из указанного набора данных - таблицы или запроса, а при использовании **Chart** данные приходится вносить самостоятельно. Поместите на форму компонент **TDBChart** со страницы **Data Controls**.
34. Свойство **Align** компонента установите равным **alClient**, чтобы график занял все оставшееся место формы.
35. Теперь дважды щелкните по графику, чтобы открыть редактор серий. Все отображаемые на графике данные построены с помощью серий - объектов **Series** типа **TChartSeries**, которые являются отображением данных отдельного реального объекта. Все настройки серий можно делать как с помощью данного редактора, так и программно изменяя свойства графика. В нашем проекте нам потребуется сделать две серии: для стипендии и для новой стипендии. **Ось X** будет содержать номера зачеток, а **ось Y** - значение.
36. Нажмите кнопку **Add (добавить серию)**. У вас появится окно выбора графика.



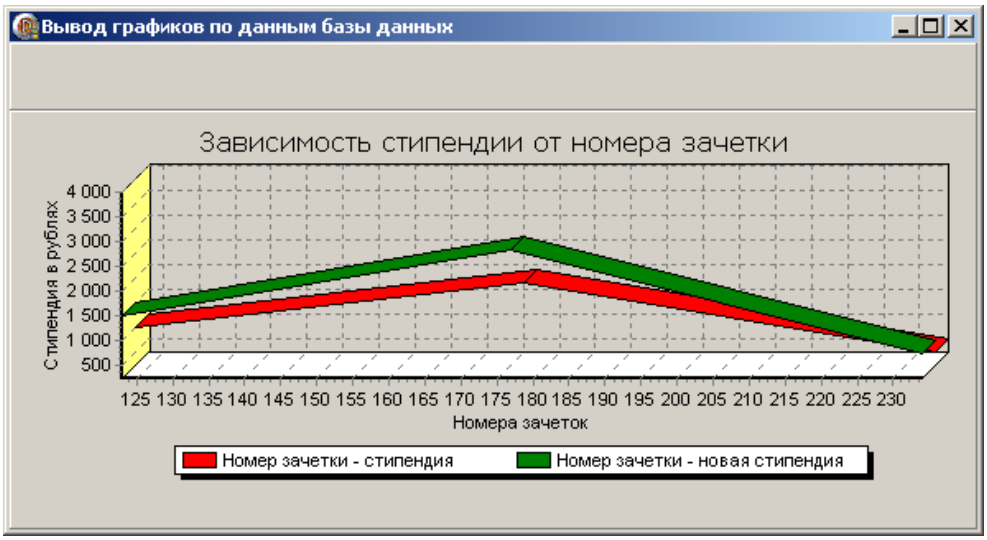
37. Помимо выбора графика мы так же можем установить флажок 3D, который включает или выключает объемность. Объемный график смотрится наряднее и больше подходит для всякого рода докладов или презентаций. Если же вам придется строить график в строгой деловой программе, то объемность будет разумней не использовать. Выбор типа графика или диаграммы зависит от типа отображаемых данных. В нашем случае удобнее выбрать тип **Line**. Выберите его. Как только это будет сделано, на компоненте **DBChart** отобразится график со случайными данными. Это происходит по умолчанию, чтобы легче было производить настройки серии.
38. В редакторе серий появилась серия **Series1**. Выделите ее и щелкните по кнопке **Title** (заголовок). Измените заголовок на **Номер зачетки – Стипендия**.
39. Теперь перейдите на вкладку **Series** в окне редактора, на этой вкладке откройте внутреннюю вкладку **Data Source**. В выпадающем списке вы видите **Random Values** (случайные значения), которые и обеспечили показ серии на графике. Нам нужно выбрать **Dataset**, а в окне **Dataset** - наш набор данных **Form1.Table1** В списке **Labels** можно выбрать поле с данными по размеру стипендии, а можно оставить его пустым, этот список используется для отображения меток, если они установлены. В списке **X** выберите поле с номером зачеток. А по оси **Y** отобразим поле с размером стипендии. Как только вы закроете редактор кнопкой **Close**, на форме появится график зависимости размера стипендии от номера зачетки.



40. Далее перейдите на вкладку **Chart** и добавьте еще одну серию. Сделайте все точно также, только отобразите пересчитанную стипендию.
41. Откройте редактор графика и перейдите на вкладку **Chart,** а в ней на вкладку **Titles**. В выпадающем списке мы видим **Title** (заголовок графика), а в текстовом окне отображается название графика **TDBChart**. Впишите вместо него **Зависимость стипендии от номера зачетки**. Кнопка **Font** позволяет изменить шрифт заголовка, кнопка **Border** откроет окно, в котором можно настроить обрамление. Кнопка **Back Color** открывает диалог выбора цвета для фона заголовка, кнопка **Pattern** также позволяет настроить фон, придав ему цвет самого графика. Если вы откроете выпадающий список, то увидите, что помимо **Title** (заголовка) доступен еще и **Foot** (подвал) - надпись, которая будет выведена внизу.
42. Вкладка **Paging** позволяет настроить многостраничные графики, а вкладка **Panel** - задать параметры фона. Интересны здесь параметры панели **Gradient**, позволяющие задать градиентную заливку. При этом фон будет плавно переходить из одного цвета в другой.
43. Вкладка **Legend** позволяет настроить легенду графика. Установите положение легенды в нижней части компонента, для этого в группе переключателей **Position** установите переключатель **Bottom**. Для параметра **Legend Style** выберите из списка значение **Automatic**.
44. Перейдите на вкладку **Axis (оси)**. Здесь мы можем сделать множество настроек осей. Вначале в левой части окна в разделе **Axis** нужно выбрать ось. Мы выберем **Left,** то есть,

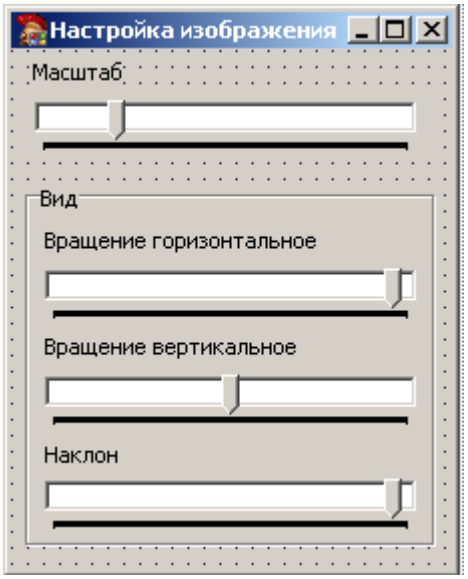
ось **Y**. Правее находится дополнительное окно со своими собственными вкладками, причем открыта вкладка **Scales** (шкалы):

45. Здесь мы снимем флажок **Automatic**, который автоматически устанавливает размер шкалы. В большинстве случаев этого не требуется, но в нашем примере мы получили относительно ровные линии, причем одна из них в нижней, а другая - в верхней части, что не делает график нагляднее. Итак, снимите эту галочку, а затем с помощью кнопок **Change** немного увеличьте максимальное значение, и немного уменьшите минимальное. В результате кривые графика сдвинутся к середине.
46. Далее можете перейти на внутреннюю вкладку **Title**, где напишите **Стипендия в рублях**. Эта надпись является заголовком оси Y.
47. В группе переключателей **Axis** выберите **Bottom** (нижняя ось), и перейдите на внутреннюю вкладку **Labels**. В разделе **Style** вместо **Auto** выберите **Value**, что изменит надписи к точкам оси X: вместо назначаемых автоматически, мы четко указали, что нужно взять значение поля. В результате этих манипуляций мы получим уже достаточно наглядный график.



48. Сохраните проект, скомпилируйте и загрузите полученную программу.
49. Откройте окно с графиками. Выделите произвольный участок графика левой кнопкой мыши слева - направо и сверху - вниз, выделенный фрагмент увеличится во все окно графика. Переместите график правой кнопкой мыши. Выделите левой кнопкой произвольный участок графика снизу - вверх и справа - налево, и масштаб графика восстановится.

50. Создайте в проекте новое окно для настройки графика. Для этого создайте новую форму и разместите на ней компоненты **TCheckBox**, **TTrackBar, TGroupBox** и **TLabel** в соответствии с рисунком.



51. Сохраните созданную форму **Form3**. Для того чтобы из окна с графиками можно было вызывать окно с настройками, а из окна настроек менять параметры графика, оба модуля придется подключить друг к другу.
52. Выполните настройку компонентов формы **Form3**. Выделите компонент **TrackBar1.** Нормальным масштабом является число **100**. Пользователю можно дать возможность изменять масштаб от **1** до **500**, еще больший масштаб будет уже неудобным. Установите свойство **Max** равным 500, свойство **Min** равным 1, а свойство **Position** равным 100. Перейдите на вкладку **Events** и для события **onChange** данного элемента управления вставьте следующий код:

    Form2.DBChart1.View3DOptions.Zoom :=TrackBar1.Position;
53. Многие возможности управления графиком зависят от того, включено ли свойство **Orthogonal**. При включенной ортогональности многие свойства становятся недоступными. Изменить состояние этого свойства можно просто:

    Form2.DBChart1.View3DOptions.Orthogonal:=False;

    Если ортогональность будет включена, пользователь сможет менять только масштаб графика, попытки изменить параметры вида графика ни к чему не приведут.
54. Выделите компонент **TrackBar2**. Горизонтальное вращение графика может быть целым числом от 0 до 360, это число указывает количество градусов. Установите свойство **Max** равным 360, свойство **Min** равным 0, а свойство **Position** равным 345. Перейдите на

вкладку **Events** и для события **onChange** данного элемента управления вставьте следующий код:

Form2.DBChart1.View3DOptions.Orthogonal:=False;

Form2.DBChart1.View3DOptions.Rotation:=TrackBar2.Position;

55. Выделите компонент **TrackBar3**. Вертикальное вращение графика может быть целым числом от 0 до 360, это число указывает количество градусов. Установите свойство **Max** равным 360, свойство **Min** равным 0, а свойство **Position** равным 180. Перейдите на вкладку **Events** и для события **onChange** данного элемента управления вставьте следующий код:

    Form2.DBChart1.View3DOptions.Orthogonal:=False;

    Form2.DBChart1.View3DOptions.Tilt:=TrackBar3.Position;

56. Выделите компонент **TrackBar4**. Наклон графика может быть целым числом от 0 до 360, это число указывает количество градусов. Установите свойство **Max** равным 360, свойство **Min** равным 0, а свойство **Position** равным 345. Перейдите на вкладку **Events** и для события **onChange** данного элемента управления вставьте следующий код:

    Form2.DBChart1.View3DOptions.Orthogonal:=False;

    Form2.DBChart1.View3DOptions. Elevation:=TrackBar4.Position;

57. Сохраните проект, скомпилируйте и загрузите полученную программу.

Задание 2. Методы команды меню «Вычислить»

Методические указания по выполнению задания:

1. Результаты вычислений выводятся на метку **Label1**. Для выполнения команды **Вычислить: - Сумму стипендий** – формирования и вывода на метку суммы столбца стипендий – используется непосредственный доступ к записям базы данных.
2. В методе данного пункта главного меню вставьте следующий программный код:

```
procedure TForm1.N19Click(Sender: TObject);
var St:string; // строка для вывода суммы стипендий
S:real;
begin
Table1.First; // перейти к первой записи
S:=0;
While Not (Table1.Eof) do // до конца файла
begin
S:=S+Table1RS.Value; // сумма стипендий
Table1.Next;        // переход к следующей записи
end;
St:=FloatToStr(S);     //  формирование строки
Label1.Caption:='Сумма стипендий='+St;       // вывод сообщения
Label1.Visible:=True;
end;
```

3. Для формирования и вывода на метку **Label1** значения средней стипендии из метода данного пункта меню вызывается метод пункта меню **Вычислить: – Сумму стипендий**. С его помощью определяется **S** - сумма стипендий, а затем её среднее. Вставьте в заготовку процедуры следующий программный код:

```
procedure TForm1.N20Click(Sender: TObject);
Var
ST: string;
N: integer;
S:real;
begin
N:=Table1.RecordCount;
N19Click(Sender);
S:=(S/N);
ST:= FloatToStr(S);
Label1.Caption:='Средняя степендия='+ST;
Label1.Visible:=True;
end;
```

4. Скрыть метку с сообщением можно щелчком по ней (метод **Label1Click**): **Label1.Visible:=False;**
5. Проверьте работоспособность проекта и сохраните изменения.

Внеаудиторная самостоятельная работа:

Составьте опорный конспект по основным командам.

Практическая работа №7

Разработка приложения для соединения данных двух таблиц 1:1. Методы объединения данных двух таблиц.

Цель работы: сформировать умения по разработке приложения, осуществляющего выбор данных из двух таблиц, не связанных на этапе разработки приложения; сформировать умения по реализации связи таблиц «один-к-одному».

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **Paradox**.

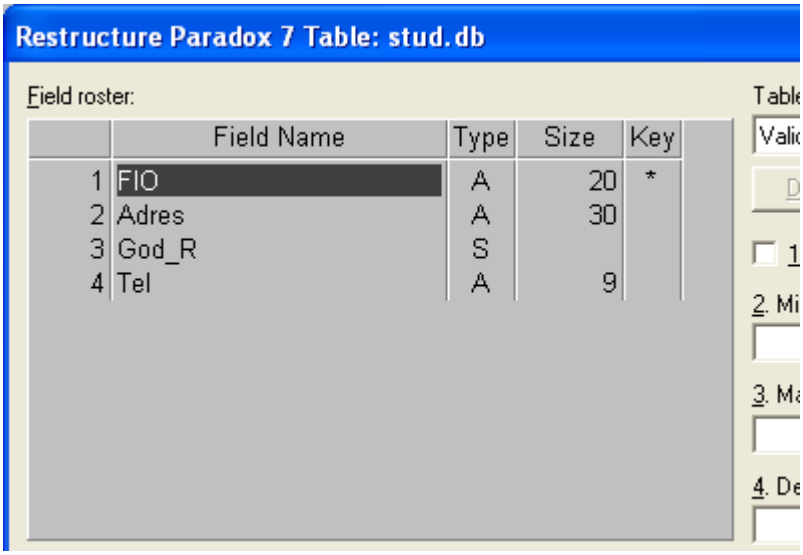
Задание.

1. Разработать базу данных в СУБД **Paradox** для хранения информации о студентах, которая состоит из двух таблиц: сведения о студентах, сведения об успеваемости студентов.
2. Разработать приложение, с помощью которого осуществляется выбор данных из двух таблиц, не связанных на этапе разработки приложения. Для разработки приложения использовать интегрированную среду разработчика **Turbo Delphi.**

Методические указания по выполнению задания:

3. В папке своей группы создайте каталог **SV_STUD**, в котором будет храниться наше приложение. В каталоге **SV_STUD** создайте подкаталог **BASE**, в котором будет хранится база данных.
4. Запустите систему **Turbo Delphi**.
5. С помощью подсистемы **Database Desktop** создайте псевдоним новой базы данных (**STUS**) и файл конфигурации.
6. Разработай структуру полей записи для двух таблиц базы данных, для этого в утилите **Database Desktop** выполните команду **File – New – Table** и перейдите к созданию первой таблицы. Она будет содержать информации о студентах. Состав поле таблицы **stud.db** представлен ниже:

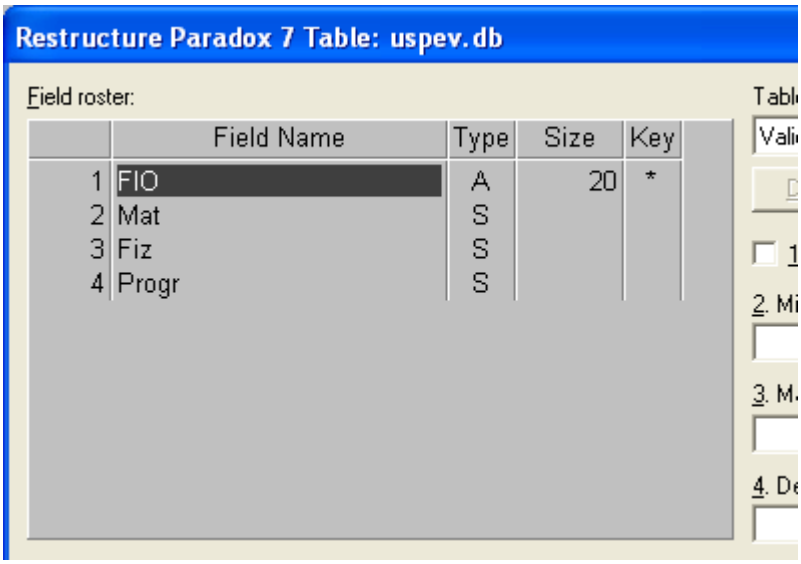
Название поля	**Описание**	**Тип данных**
FIO	Фамилия и инициалы	Текстовый, размер 20, ключевое поле
Adres	Домашний адрес	Текстовый, размер 30
God_R	Год рождения	Целое число
Tel	Телефон	Текстовый, размер 9



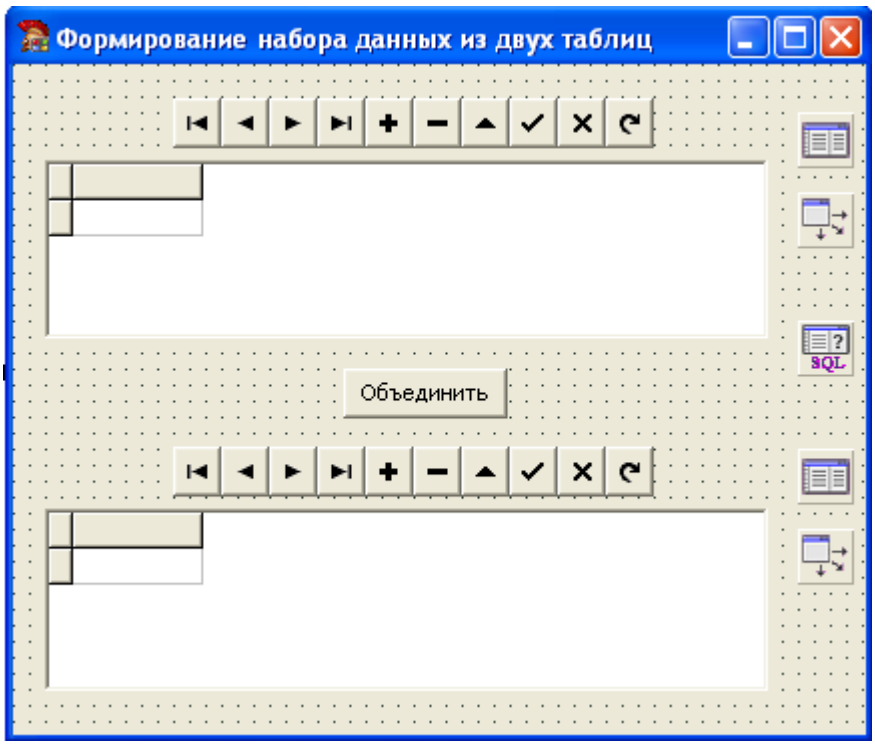
7. Создайте вторую таблицу базы данных. Она будет содержать данные о успеваемости студентов. Состав полей таблицы **uspev.db** представлен ниже:

Название поля	**Описание**	**Тип данных**
FIO	Фамилия и инициалы	Текстовый, размер 20, ключевое поле

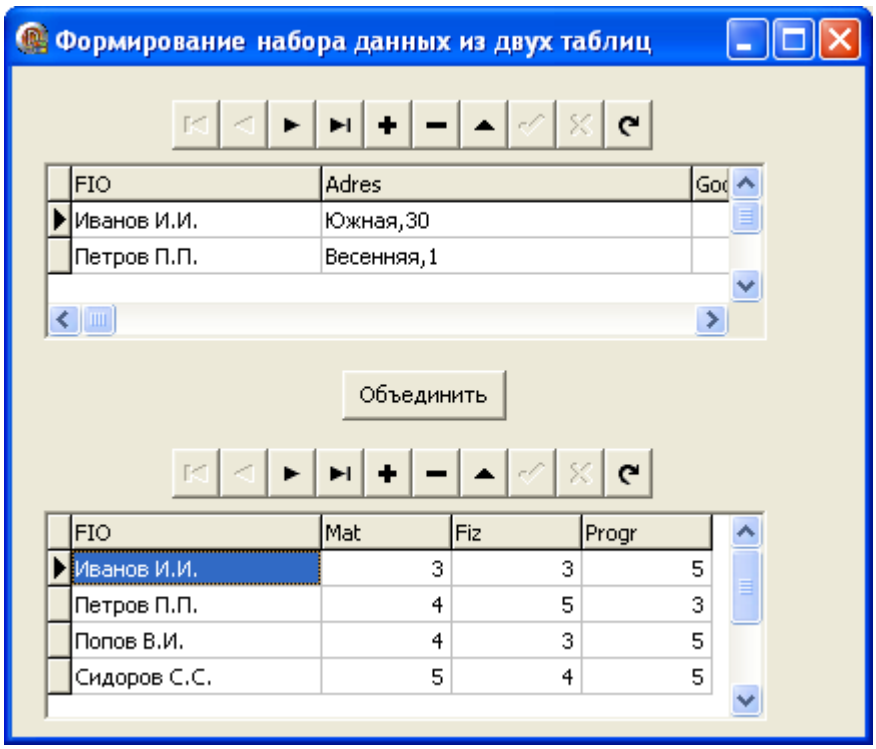
Mat	Оценка по математике	Целое число
Fiz	Оценка по физике	Целое число
Progr	Оценка по программированию	Целое число



8. Перейдите в среду разработки приложений **Turbo Delphi**. Создайте новый проект, выполнив команду **File – New – VCL Forms Application**. Установите свойство формы **Caption** равным **Формирование набора данных из двух таблиц.**
9. Нанесите на форму следующие компоненты для работы с базами данных: два компонента типа **Table**, два компонента типа **DataSource**, два компонента **DBGrid**, два компонента **DBNavigator**, компонент **Query**, компонент типа **Button**. Размещение компонентов на форме представлено на рисунке.



10. Установите для компонентов следующие свойства:
    - **Table1: DatabaseName=STUS, TableName=stud.db, Active=True,**
    - **Table2: DatabaseName=STUS, TableName=uspev.db, Active=True,**
    - **Query1: DatabaseName=STUS, DataSource – пустая строка,**
    - **DataSource1: DataSet=Table1,**
    - **DataSource2: DataSet=Table2,**
    - **DBGrid1** и **DBNavigator1: DataSource= DataSource1,**
    - **DBGrid2** и **DBNavigator2: DataSource= DataSource2**
11. В процессе настройки компонентов **DBGrid** не изменяйте заголовки столбцов, установленные по умолчанию.
12. Отбор данных из двух таблиц осуществляется с помощью компонента **Query1**. Для этого установите его свойство **SQL** равным **SELECT * FROM STUD, USPEV WHERE STUD.FIO=USPEV.FIO** В соответствии с ним в одну строку формируемого набора данных будут отображаться значения из первой и второй таблиц, у которых совпадают значения полей с фамилиями.
13. Запустите приложение и заполните таблицы несколькими записями с помощью компонентов **DBNavigator1** и **DBNavigator2**.

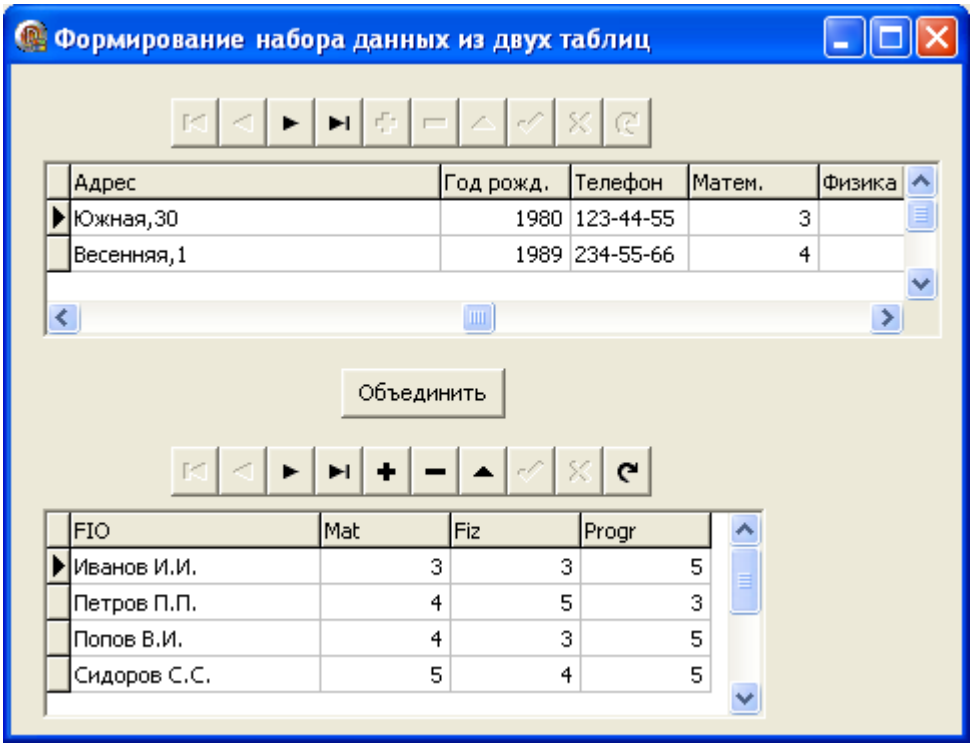


14. С помощью метода **Button1Click** нужно определить типизированную константу **C** из массива строк с наименованиями заголовков компонента **DBGrid1**, в котором будут отражены значения полей отобранных и сформированных строк данных; активизировать компонент **Query1;** подключить **Query1** к компоненту **DataSource1** для отображения отобранных данных; вывести тексты заголовков столбцов таблицы из массива **C**.
15. Реализуем данный алгоритм для этого выполните двойной щелчок по кнопке **Объединить** и в окне заготовки процедуры запишите следующий программный код:

```
procedure TForm1.Button1Click(Sender: TObject);
Const C: Array [0..7] of string[10]=
('ФИО', 'Адрес', 'Год рожд.', 'Телефон', '', 'Матем.', 'Физика', 'Прогр.');
var I:byte;
begin
IF Query1.Active=False THEN Query1.Active:=True;
DataSource1.DataSet:=Query1;
DBGrid1.Columns[4].Visible := False;
FOR I:=0 to 7 DO
DBGrid1.Columns[I].Title.Caption :=C[i];
end;
```

16. Сохраните проект, скомпилируйте его и запустите.
17. Вид формы после объединения данных двух таблиц на этапе выполнения приложения представлен на рисунке ниже. В нижней таблице в каждой строке представлены данные

из первой и второй таблиц о каждом студенте, фамилия которого есть и в первой и во второй таблице.



Внеаудиторная самостоятельная работа:

Ответьте на вопросы письменно в тетради:

- Поясните текст свойства SQL для отбора данных из двух несвязанных таблиц.
- Поясните текст метода для установки заголовков таблиц в процессе выполнения приложения.
- Как сделать невидимым один из отобранных столбцов результирующей таблицы?

Практическая работа №8

Разработка приложения для формирования, слияния и разъединения однотипных таблиц баз данных. Основные методы и свойства компонента BatchMove.

На этом этапе осуществляется *отображение* элементов полученных ранее моделей классов в элементы моделей *базы данных* и приложений:

- классы отображаются в таблицы;
- атрибуты – в столбцы;
- типы – в типы данных используемой СУБД;
- ассоциации – в связи между таблицами (ассоциации "многие-ко-многим" преобразуются в ассоциации "один-ко-многим" посредством создания дополнительных таблиц связей);

- приложения – в отдельные классы с окончательно определенными и связанными с данными в базе методами и атрибутами.

Поскольку модели *базы данных* и приложений строятся на основе единой логической модели, автоматически обеспечивается *связность* этих проектов (рис. 1).

Рис. 1. Связь между проектами базы данных и приложений

В модель *базы данных* отображаются только перманентные классы, из которых удаляются атрибуты, не отображаемые в столбцах (например, *атрибут* типа " Общий объем продаж ", который получается суммированием содержимого *множества* полей *базы данных*). Некоторые атрибуты (например, АДРЕС) могут отображаться в множество столбцов (СТРАНА, ГОРОД, УЛИЦА, ДОМ, ПОЧТОВЫЙ ИНДЕКС).

Для каждого простого класса в модели *базы данных* формируется отдельная *таблица*, включающая столбцы, соответствующие атрибутам класса.

Отображение классов подтипов в таблицы осуществляется одним из стандартных способов:

- одна таблица на класс;
- одна таблица на суперкласс;
- одна таблица на иерархию.

В первом случае для каждого из классов создается отдельная *таблица*, между которыми затем устанавливаются необходимые связи. Во втором случае создается *таблица* для суперкласса, а затем в каждую таблицу подклассов включаются столбцы для каждого из атрибутов суперкласса. В третьем – создается единая *таблица*, содержащая атрибуты как суперкласса, так и всех подклассов (рис. 12.13). При этом для выделения исходных таблиц подклассов в результирующую таблицу добавляется один или более дополнительных столбцов (на рисунке показан курсивом).

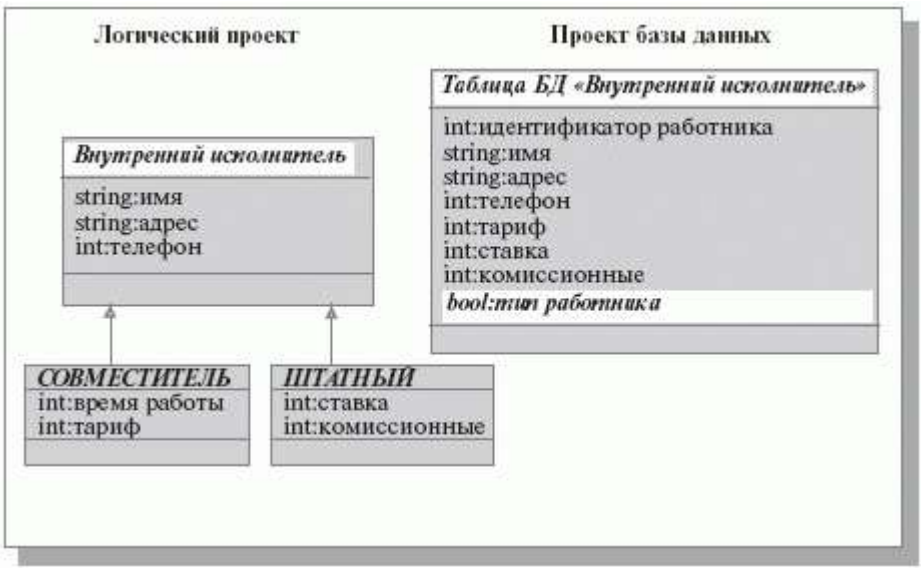


Рис. 2. Преобразование иерархии в таблицу

Разработка проекта *базы данных* осуществляется с использованием специального *UML*-профиля (Profile for *Database* Design), который включает следующие основные компоненты диаграмм:

- таблица – набор записей базы данных по определенному объекту;
- столбец – элемент таблицы, содержащий значения одного из атрибутов таблицы;
- первичный ключ (PK) – атрибут, однозначно идентифицирующий строку таблицы;
- внешний ключ (FK) – один или группа атрибутов одной таблицы, которые могут использоваться как первичный ключ другой таблицы;
- обязательная связь – связь между двумя таблицами, при которой дочерняя таблица существует только вместе с родительской;
- необязательная связь – связь между таблицами, при которой каждая из таблиц может существовать независимо от другой;
- представление – виртуальная таблица, которая обладает всеми свойствами обычной таблицы, но не хранится постоянно в базе данных;
- хранимая процедура – функция обработки данных, выполняемая на сервере;
- домен – множество допустимых значений для столбца таблицы.

На рис. 3 представлен фрагмент модели *базы данных* — две таблицы, соответствующие классам " пациент " (рис. 2) и " минимальный набор данных ". *Связь* между ними обязательная, поскольку " минимальный набор данных " не может существовать без " пациента ".

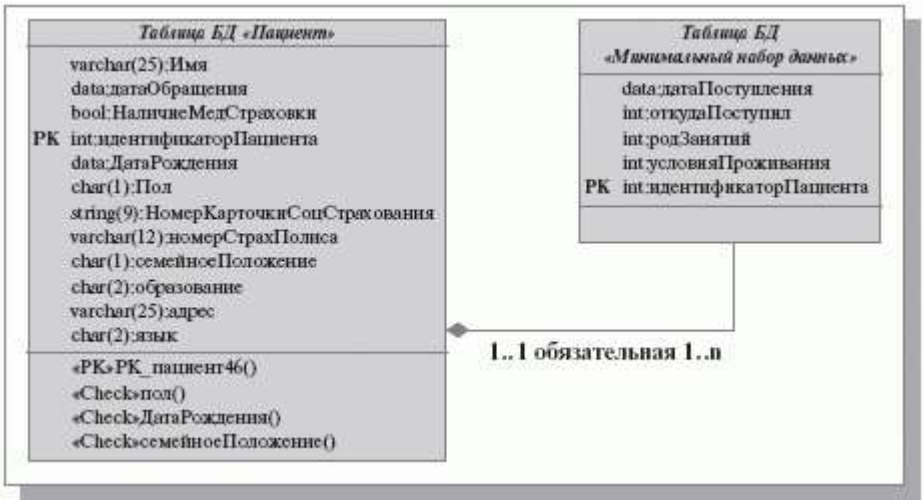


Рис. 3. Фрагмент модели базы данных

На диаграммах указываются дополнительные характеристики таблиц и столбцов:

- ограничения – определяют допустимые значения данных в столбце или операции над данными (ключ (PK,FK) – ограничение, определяющее тип ключа и его столбец; проверка (Check) – ограничение, определяющее правило контроля данных; уникальность (Unique) – ограничение, определяющее, что в столбце содержатся неповторяющиеся данные);
- триггер – программа, выполняющая при определенных условиях предписанные действия с базой данных;
- тип данных и пр.

Результатом этапа является детальное описание проекта *базы данных* и приложений системы.

Практическая работа №9

Разработка приложения с таблицей для выбора допустимых значений. Установка связи головной и вспомогательной таблиц при создании БД в DatabaseDesktop. Поля просмотра lookupfields. Использование редактора полей при создании нового поля зависимой таблицы. Разработка приложения для таблиц, связанных с помощью свойства ReferentialIntegrity.

Цель работы: изучить особенности организации приложения с таблицей для выбора допустимых значений с помощью свойства Lockup, изучить особенности организации приложения с таблицей для выбора допустимых значений с помощью нового поля таблицы с выпадающим списком, изучить особенности создания связи между таблицами с помощью свойства **Referential Integrity** подчиненной таблицы и выпадающего списка допустимых значений с помощью ее свойства **Lookup,** сформировать умения по разработке приложения, с помощью которого осуществляется работа с данными двух связанных таблиц.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **Paradox**.

Задание 1. Разработка приложения с таблицей для выбора допустимых значений с помощью свойства Lockup

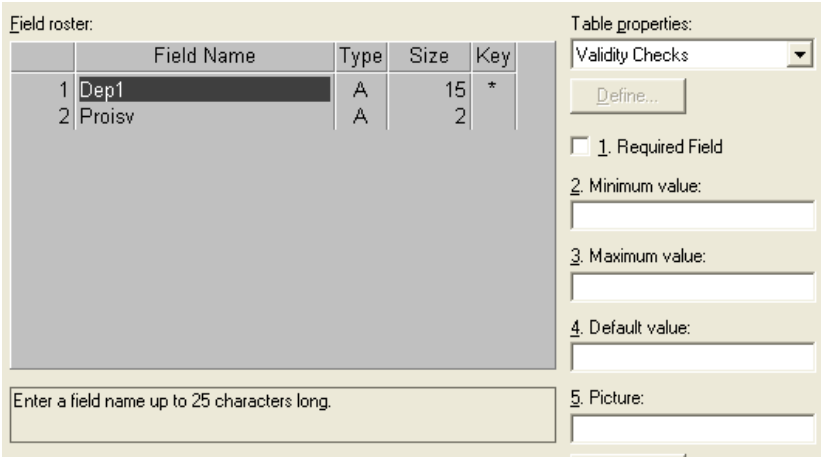
Методические указания по выполнению задания:

1. В папке своей группы создайте каталог **LOOKUP1**, в котором будет храниться наше приложение. В каталоге **LOOKUP1** создайте подкаталог **BASE**, в котором будет храниться база данных.
2. Запустите систему **Turbo Delphi**.
3. С помощью подсистемы **Database Desktop** создайте псевдоним новой базы данных (**Dep**) и файл конфигурации.
4. Требуется разработать приложение с двумя зависимыми таблицами. В каждой таблице есть поле **Dep** – наименование подразделения. Основная таблица **Dep**. Она содержит в поле **Dep1** список имеющихся подразделений организации. **Pers** – подчиненная (зависимая) таблица. В поле **Dep2** таблицы **Pers** сформирован выпадающий список допустимых значений – из поля **Dep1** таблицы **Dep**. При заполнении очередной записи

в таблице **Pers** следует произвольно заполнять все её поля, кроме поля **Dep2**. При заполнении поля **Dep2** таблицы **Pers** надо использовать значения из выпадающего списка допустимых значений. Связь между таблицами устанавливается при разработке структуры зависимой таблицы. Для этого нужно разработать структуру записи основной таблицы **Dep** и структуру записи таблицы **Pers**.

5. Разработайте структуру полей записи таблицы **Dep** базы данных, для этого в утилите **Database Desktop** выполните команду **File – New – Table** и перейдите к созданию первой таблицы. Она будет содержать информации о студентах. Состав поле таблицы **Dep.db** представлен ниже:

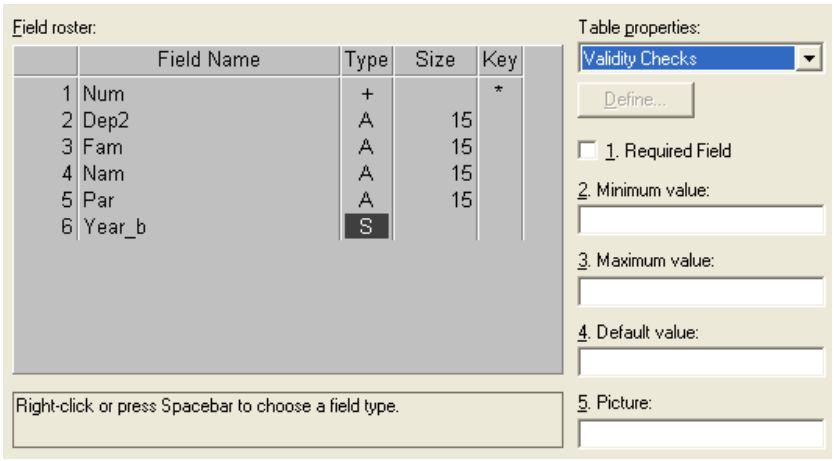
Название поля	Описание	Тип данных
Dep1	Наименование отдела	Текстовый, размер 15, основной ключ
Proisv	Признак типа отдела	Текстовый, размер 2



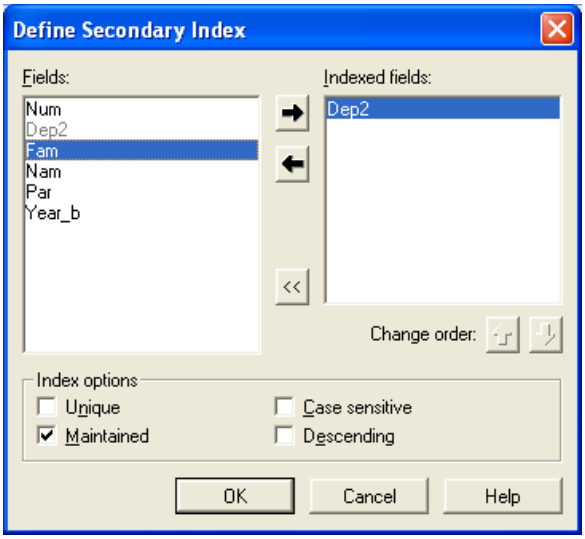
6. Разработайте структуру полей записи таблицы **Pers** базы данных, для этого в утилите **Database Desktop** выполните команду **File – New – Table** и перейдите к созданию первой таблицы. Она будет содержать информации о студентах. Состав поле таблицы **Pers.db** представлен ниже:

Название поля	Описание	Тип данных
Num	Порядковый номер записи	Счетчик, основной ключ
Dep2	Наименование отдела	Текстовый, 15
Fam	Фамилия	Текстовый, 15
Nam	Имя	Текстовый, 15

Par	Отчество	Текстовый, 15
Year_v	Год рождения	Числовой
Sex	Пол	Логический

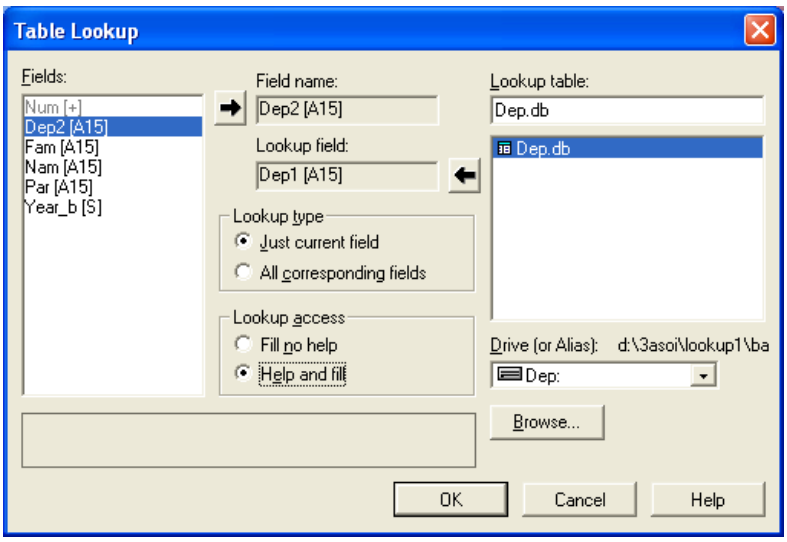


7. Для таблицы **Pers** установите в качестве вторичного индекса поле **Dep2**. Для этого в окне **Create Paradox Table** из выпадающего списка с заголовком **Table Properties** выберите параметр **Secondary Indexes**. Индексы используются при поиске данных для сортировки записей по возрастанию и убыванию значений столбца, для которого сформирован вторичный индекс. При задании индексов надо определить состав полей индекса, его параметры и задать его имя. После выбора этого параметра из списка свойств появляются кнопки **Define (Определить)**, **Modify (Модифицировать)**, **Erase (Удалить)**. Для создания нового индекса нужно нажать кнопку **Define**. Появится окно **Define Secondary Index**, в нем задается состав полей и параметры индекса. В состав одного индекса может входить одно и более полей записи таблицы. В списке **Fields** окна выводятся все поля записи таблицы. В список **Indexed fields** с помощью стрелки копируются имена полей записи, которые должны войти в состав создаваемого индекса. Кнопки с горизонтальными стрелками между списками позволяют включить или исключить поле из состава индекса.

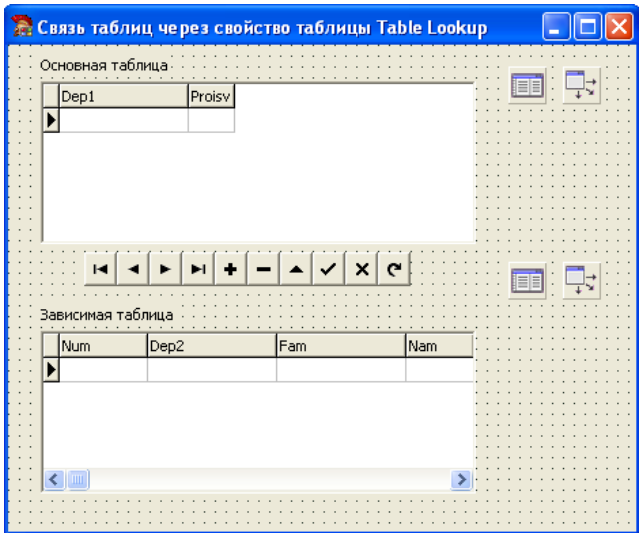


8. После определения нового индекса и нажатия кнопки **ОК** появляется окно **Save Index As**, в поле **Index Name** нужно ввести имя созданного индекса **Ind_Dep**. После нажатия кнопки **ОК** сформированный индекс добавляется к таблице, и его имя появляется в списке созданных индексов.
9. Для установки связи полей **Dep1** и **Dep2** основной и зависимой таблиц нужно при разработке структуры записи зависимой таблицы вызвать список свойств таблицы **Table Properties**. Из него выбрать пункт **Table Lookup**. Под списком становится доступной кнопка **Define**. После нажатия на эту кнопку появляется окно **Table Lookup**.
10. В списке **Fields** выведены имена всех полей зависимой таблицы **Pers.** Имена полей, которые не допускают создание таблицы выбора, выделены серым цветом. Для установки зависимости поля **Dep2[A15]** таблицы надо выделить его в списке **Fields** и нажать кнопку со стрелкой вправо. Имя этого поля перейдет в поле с заголовком **Field Name**.
11. Затем надо в поле **Lookup table** ввести имя главной (основной) таблицы. Для этого надо в поле **Drive (or Alias)** выбрать из списка псевдоним базы данных **Dep**. После задания таблицы для выбора надо нажать кнопку со стрелкой влево. При этом имя первого попавшегося поля с типом параметра **Dep2[A15]**, совпадающего с типом параметра в окне **Field name Dep1[A15]**, копируется в поле окна с заголовком **Lookup field.**
12. В группе переключателей **Lookup type** можно задать тип выбора. Для этого выберите **Just current field** – Только текущее поле. В группе **Lookup access** оставьте включенным доступ в виде **Help and fill**. Этот параметр действует только при работе в среде **Desktop**.

В среде **Delphi** список допустимых значений создается программно. Щелкните по кнопке **ОК**.



13. Перейдите в среду разработки приложений **Turbo Delphi**. Создайте новый проект, выполнив команду **File – New – VCL Forms Application**. Установите свойство формы **Caption** равным **Связь таблиц через свойство таблицы Table Lookup**.
14. Нанесите на форму следующие компоненты для работы с базами данных: два компонента типа **Table**, два компонента типа **DataSource**, два компонента **DBGrid**, один компонент **DBNavigator**, два компонента типа **Label**. Размещение компонентов на форме представлено на рисунке.

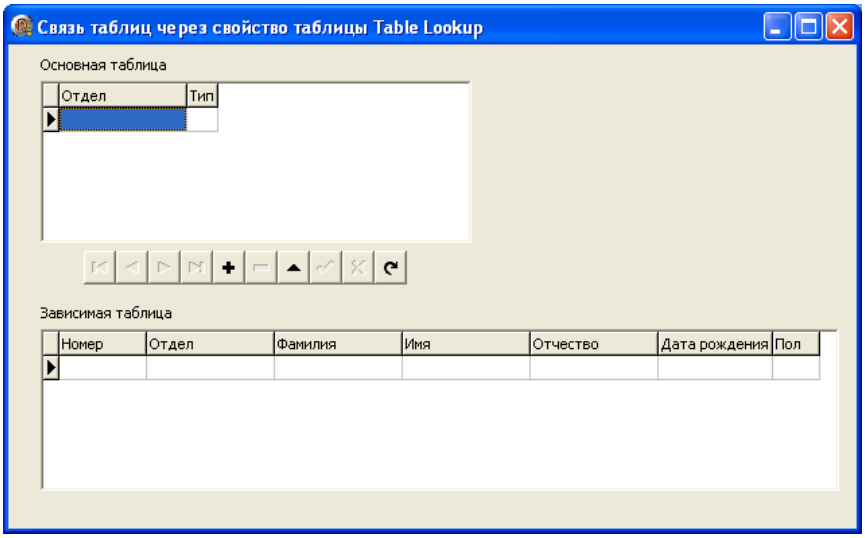


18. Установите для компонентов следующие свойства:

- **Label1: Caption=Основная таблица;**
- **Label2: Caption=Зависимая таблица;**

- **Table1: DatabaseName=DEP, TableName=Dep.db, IndexFieldNames=Dep1, Active=True,**
- **Table2: DatabaseName=DEP, TableName=Pers.db, IndexFieldNames=Num, Active=True, MasterSource=DataSource1,**
- **DataSource1: DataSet=Table1,**
- **DataSource2: DataSet=Table2,**
- **DBGrid1** и **DBNavigator1: DataSource= DataSource1,**
- **DBGrid2: DataSource= DataSource2**

15. Установите русские заголовки столбцов в компонентах **DBGrid1** и **DBGrid2**.



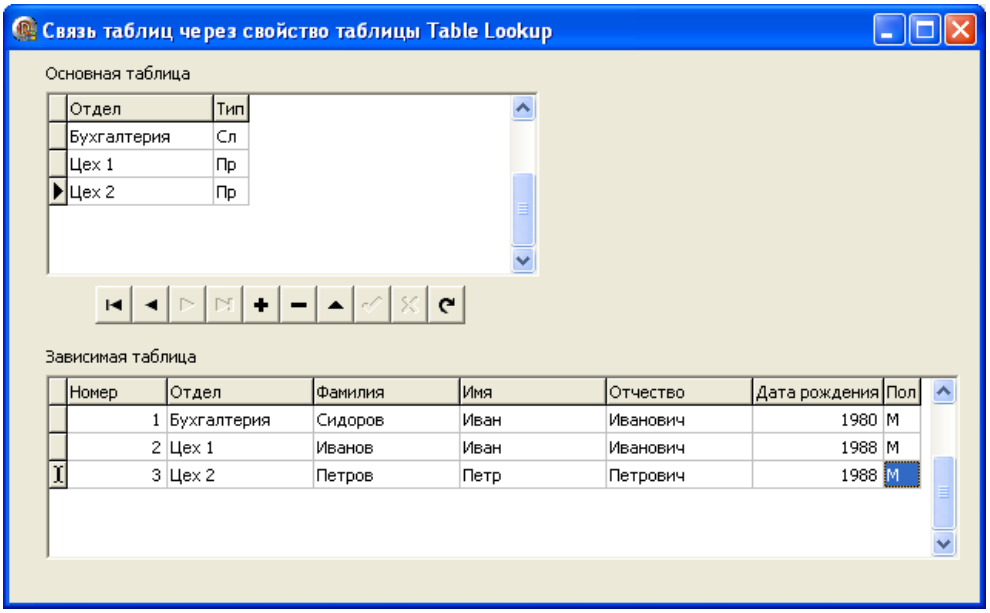
16. Далее настроим выпадающий список второго столбца компонента **DBGrid2**. Для этого вызовите редактор **Editing DBGrid2.Columns**. В нем выберите поле **Dep2** и установите его свойство **ButtonStyle=cbsAuto**. Это необходимо для того, чтобы в поле второго столбца появилась стрелка для вызова выпадающего списка допустимых значений.
17. Для вывода вместо булевых смысловых значений в таблице **DBGrid2** установите значения в свойстве **DisplayValues**. Для этого вызовите редактор таблицы **Table2**, выберите в нем поле **Sex** и установите значение поля: **DisplayValues=М,Ж**.
18. Запустите приложение и создайте несколько записей в основной таблице.
19. Формирование списка допустимых значений для поля **Dep2** таблицы **Table2** из значений поля **Dep1** таблицы **Table1** производится программно при создании формы с помощью метода **TForm1.FormCreate**. Метод обеспечивает:
    - очистку списка с помощью оператора **DBGrid2.Columns[1].PickList.Clear**
    - перебор записей **Table1** – таблицы-источника с помощью методов **First** и **Next;**

- дополнение списка в **DBGrid2.Columns[1]** очередным значением из **Table1.FieldByName('Dep1')** с помощью оператора:

DBGrid2.Columns[1].PickList.Add(Table1.FieldByName('Dep1').Value);

```
procedure TForm1.FormCreate(Sender: TObject);
begin
DBGrid2.Columns[1].PickList.Clear;
with Table1 do
  begin
    First;
    While Not Eof do
      begin
        DBGrid2.Columns[1].PickList.Add(FieldByName('Dep1').Value);
        Next;
      end;
  end;
end;
```

20. Сохраните проект, скомпилируйте его и запустите. Создайте несколько записей в зависимой таблице.



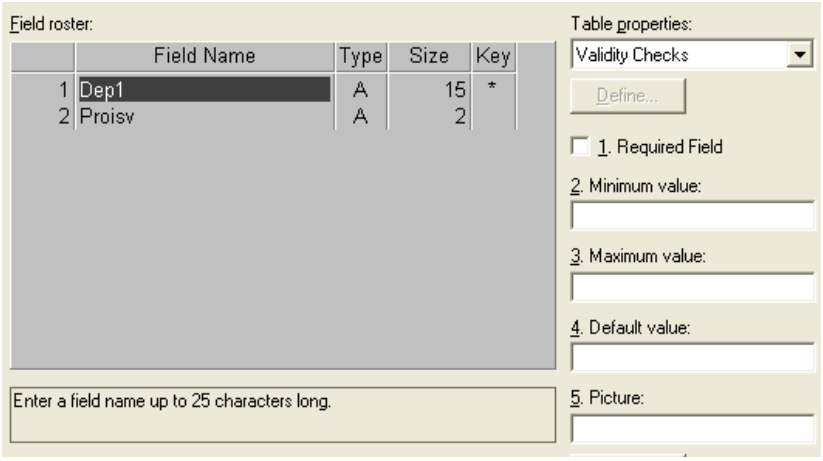
Задание 2. Разработка приложения с таблицей для выбора допустимых значений с помощью нового поля таблицы с выпадающим списком

Методические указания по выполнению задания:

1. В папке своей группы создайте каталог **LOOKUP2**, в котором будет храниться наше приложение. В каталоге **LOOKUP2** создайте подкаталог **BASE**, в котором будет храниться база данных.

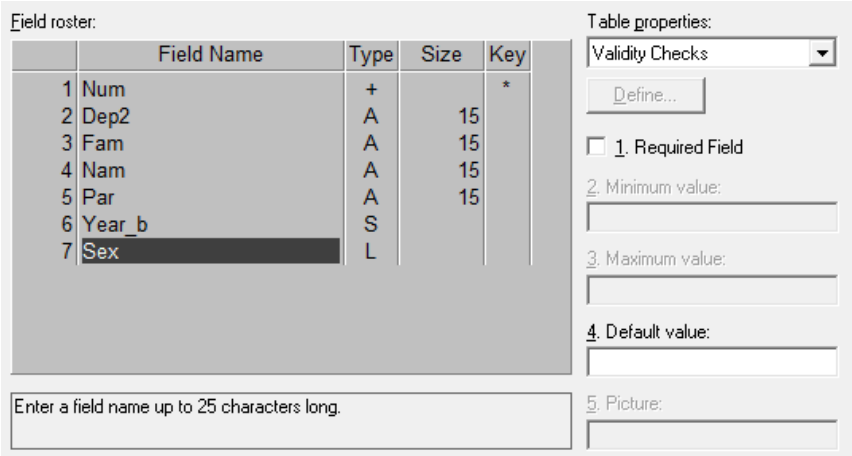
2. С помощью подсистемы **Database Desktop** создайте псевдоним новой базы данных (**Dep1**) и файл конфигурации.
3. Требуется разработать приложение с двумя таблицами: основной и зависимой. Допустимые значения для зависимой таблицы определяются из основной таблицы. Для создания списка допустимых значений использовать новое поле. Для этого нужно разработать структуру записи основной таблицы **Dep** и структуру записи таблицы **Pers.**
4. Разработайте структуру полей записи таблицы **Dep** базы данных, для этого в утилите **Database Desktop** выполните команду **File – New – Table** и перейдите к созданию первой таблицы. Состав поле таблицы **Dep.db** представлен ниже:

Название поля	**Описание**	**Тип данных**
Dep1	Наименование отдела	Текстовый, размер 15, основной ключ
Proisv	Признак типа отдела	Текстовый, размер 2

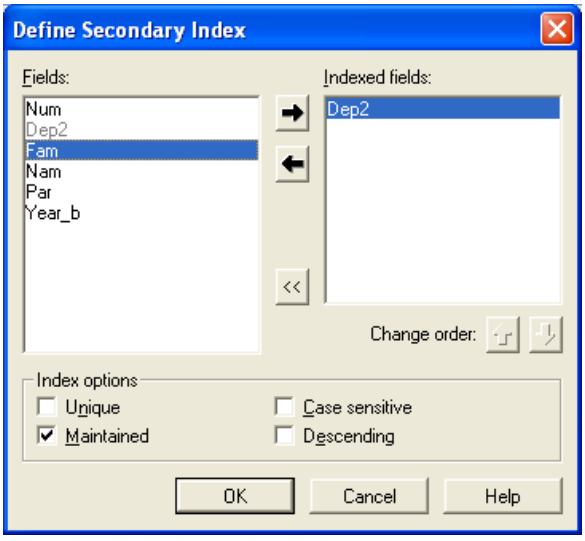


5. Разработайте структуру полей записи таблицы **Pers** базы данных, для этого в утилите **Database Desktop** выполните команду **File – New – Table** и перейдите к созданию первой таблицы. Состав поле таблицы **Pers.db** представлен ниже:

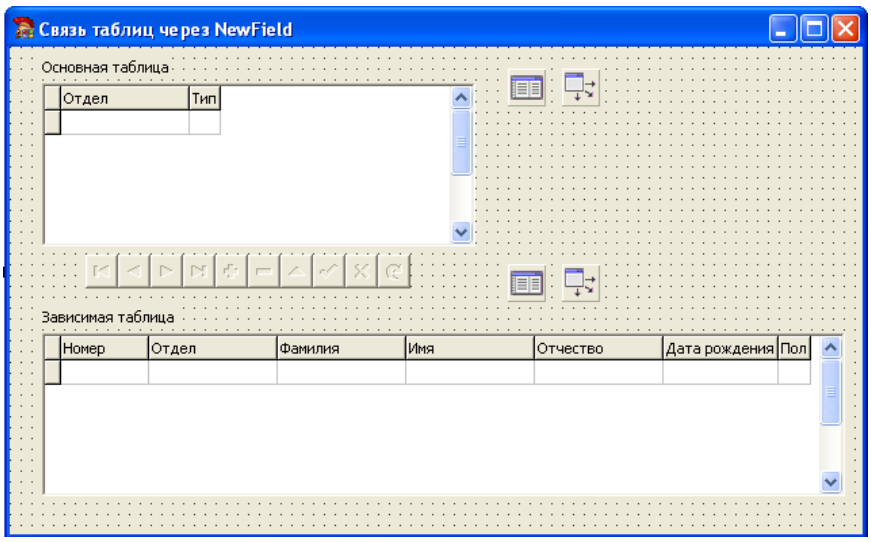
Название поля	**Описание**	**Тип данных**
Num	Порядковый номер записи	Счетчик, основной ключ
Dep2	Наименование отдела	Текстовый, 15
Fam	Фамилия	Текстовый, 15
Nam	Имя	Текстовый, 15
Par	Отчество	Текстовый, 15
Year_b	Год рождения	Числовой
Sex	Пол	Логический



6. Для таблицы **Pers** установите в качестве вторичного индекса поле **Dep2**. Для этого в окне **Create Paradox Table** из выпадающего списка с заголовком **Table Properties** выберите параметр **Secondary Indexes**. Индексы используются при поиске данных для сортировки записей по возрастанию и убыванию значений столбца, для которого сформирован вторичный индекс. При задании индексов надо определить состав полей индекса, его параметры и задать его имя. После выбора этого параметра из списка свойств появляются кнопки **Define (Определить)**, **Modify (Модифицировать)**, **Erase (Удалить)**. Для создания нового индекса нужно нажать кнопку **Define**. Появится окно **Define Secondary Index**, в нем задается состав полей и параметры индекса. В состав одного индекса может входить одно и более полей записи таблицы. В списке **Fields** окна выводятся все поля записи таблицы. В список **Indexed fields** с помощью стрелки копируются имена полей записи, которые должны войти в состав создаваемого индекса. Кнопки с горизонтальными стрелками между списками позволяют включить или исключить поле из состава индекса.



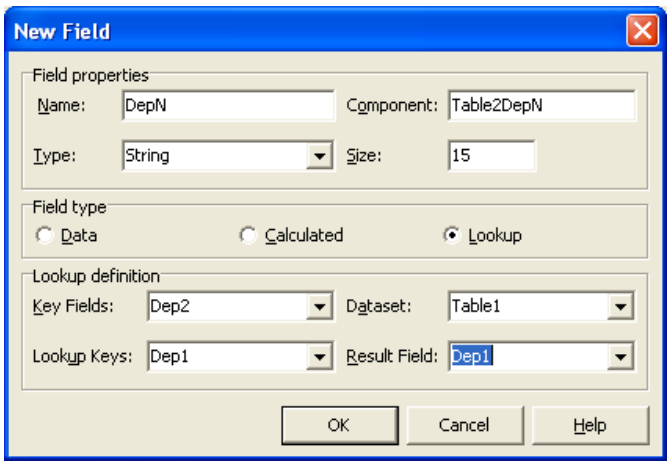
7. После определения нового индекса и нажатия кнопки **ОК** появляется окно **Save Index As**, в поле **Index Name** нужно ввести имя созданного индекса **Ind_Dep**. После нажатия кнопки **ОК** сформированный индекс добавляется к таблице, и его имя появляется в списке созданных индексов.
8. Перейдите в среду разработки приложений **Turbo Delphi**. Создайте новый проект, выполнив команду **File – New – VCL Forms Application**. Установите свойство формы **Caption** равным **Связь таблиц через NewField**.
9. Нанесите на форму следующие компоненты для работы с базами данных: два компонента типа **Table**, два компонента типа **DataSource**, два компонента **DBGrid**, один компонент **DBNavigator**, два компонента типа **Label**. Размещение компонентов на форме представлено на рисунке.
10. Установите для компонентов следующие свойства:
    - **Label1: Caption=Основная таблица;**
    - **Label2: Caption=Зависимая таблица;**
    - **Table1: DatabaseName=DEP1, TableName=Dep.db, IndexFieldNames=Dep1, Active=True,**
    - **Table2: DatabaseName=DEP1, TableName=Pers.db, IndexFieldNames=Num, Active=True, MasterSource=DataSource1,**
    - **DataSource1: DataSet=Table1,**
    - **DataSource2: DataSet=Table2,**
    - **DBGrid1** и **DBNavigator1: DataSource= DataSource1,**
    - **DBGrid2: DataSource= DataSource2**
11. Установите русские заголовки столбцов в компонентах **DBGrid1** и **DBGrid2**.



12. Запустите приложение и создайте несколько записей в основной таблице.
13. Для формирования связи между таблицами нужно вызвать редактор полей таблицы **Table2**, добавить все поля. Вызвать его контекстное меню, выбрать в нем команду **New Field**. Появится окно **New Field**. В его разделе **Field Properties** установить свойства:
    - в окне **Name:** имя нового поля - **DepN**;
    - в окне **Component:** появится имя нового поля в таблице - **Table2DepN**;
    - в окне **Type:** выбрать из выпадающего списка требуемый тип нового поля - **String**;
    - в окне **Size:** установить размер строки в символах - **15**.

В разделе **Field Type** выбрать тип **Lookup** - выбор. В разделе **Lookup Definition** (определение выбора) установить выбором из выпадающих списков значения:

- в окне **Key Fields:** ключевое поле связи зависимой таблицы (**Table2**) - **Dep2**;
- в окне **Data Set:** имя таблицы-источника допустимых значений - **Table1**;
- в окне **Lookup Keys:** ключевое поле связи исходной таблицы (**Table1**) – **Dep1**;
- в окне **Result Field:** имя поля из **Table1** для выпадающего списка в **Table2** - **Dep1**.



14. В результате формирования нового поля с выпадающим списком в редакторе таблицы **Table2** появится новое поле с его свойствами. Основные свойства поля с выпадающим списком:
    - **FieldKind = fkLookup;** - тип нового поля - выпадающее;
    - **FieldName = DepN;** - имя нового поля;
    - **KeyFields = Dep2;** - ключевое поле связи зависимой таблицы **Table2**;
    - **LookupDataSet =Table1;** - имя таблицы-источника допустимых значений;
    - **LookupKeyFields = Dep1;** - ключевое поле связи исходной таблицы - **Table1**;
    - **LookupResultField = Dep1;** - имя поля из **Table1** для выпадающего списка.

Поля связи не обязательно должны быть индексными. Поле связи (**Dep1**) не обязательно должно быть полем выпадающего списка (**Result Field = Dep1**).

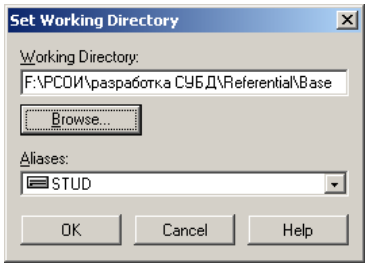
15. Значения полей **Dep2** и **DepN** идентичны. Столбец со значениями **Dep2** можно сделать невидимым для этого установите его свойство **Visible** равным **False**.
16. Сохраните проект, скомпилируйте его и запустите. Создайте несколько записей в зависимой таблице.

Задание 3. Разработка приложения для таблиц, связанных с помощью свойства Referential Integrity

Методические указания по выполнению задания:

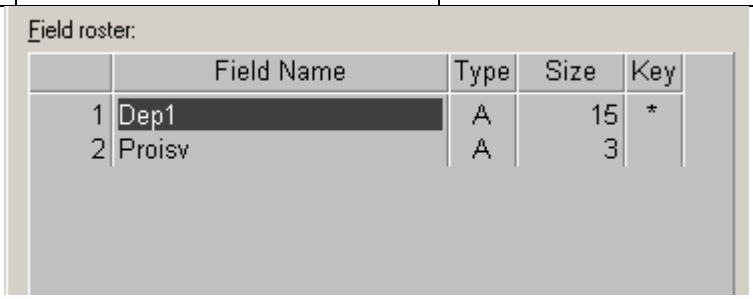
1. В папке своей группы создайте каталог **Referential**, в котором будет храниться наше приложение. В каталоге **Referential** создайте подкаталог **BASE**, в котором будет храниться база данных.
2. Запустите систему **Turbo Delphi**.

3. С помощью подсистемы **Database Desktop** создайте псевдоним новой базы данных (**Dep2**) и файл конфигурации.
4. С помощью подсистемы **Database Desktop** установите текущим каталог с базой данных. Для этого нужно в системе **Database Desktop** вызвать команду **File - Working Directory**. Появится окно для установки директории. Нажмите кнопку **Browse**, выберите диск и каталог с требуемой базой данных.



5. Разработайте структуру полей записи основной таблицы **Dep** базы данных, для этого в утилите **Database Desktop** выполните команду **File – New – Table** и перейдите к созданию первой таблицы. Состав поле таблицы **Dep.db** представлен ниже:

Название поля	**Описание**	**Тип данных**
Dep1	Наименование отдела	Текстовый, размер 15, основной ключ
Proisv	Признак типа производства	Текстовый, размер 3

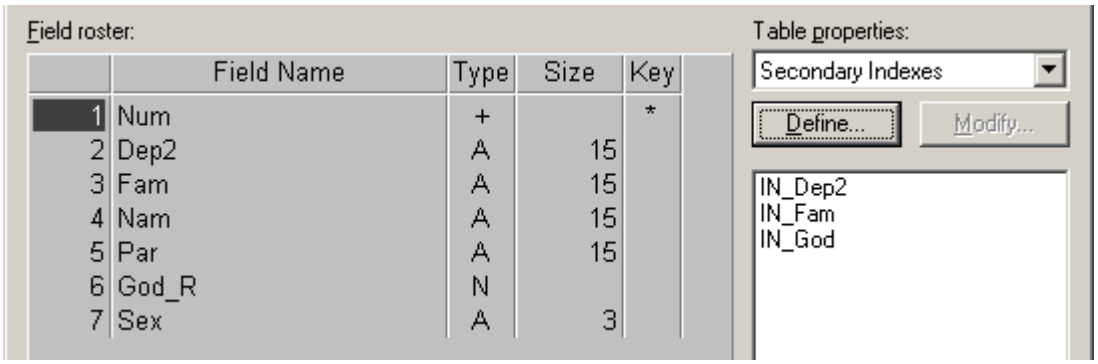


6. Разработайте структуру полей записи таблицы **Pers1** базы данных, для этого в утилите **Database Desktop** выполните команду **File – New – Table** и перейдите к созданию первой таблицы. Состав поле таблицы **Pers1.db** представлен ниже:

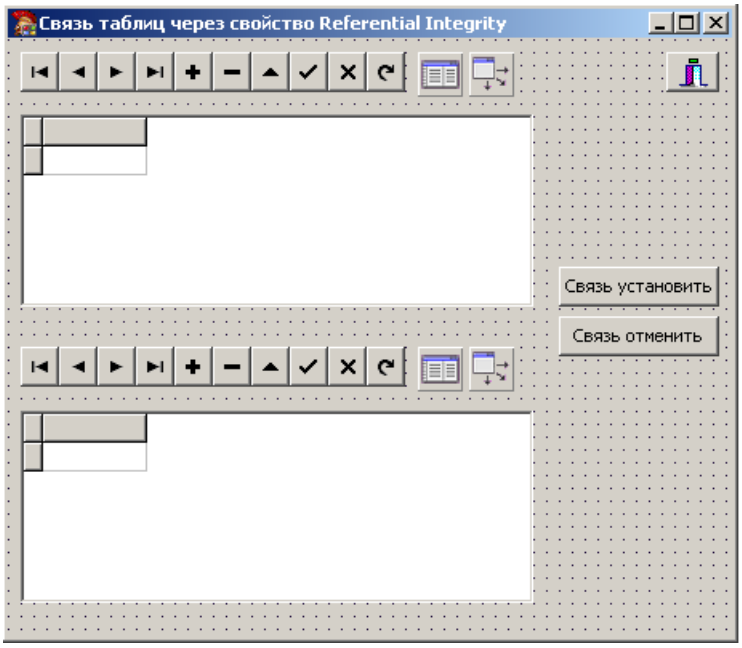
Название поля	**Описание**	**Тип данных**
Num	Порядковый номер записи	Счетчик, основной ключ
Dep2	Наименование отдела	Текстовый, 15
Fam	Фамилия	Текстовый, 15
Nam	Имя	Текстовый, 15
Par	Отчество	Текстовый, 15

God_R	Год рождения	Числовой
Sex	Пол	Текстовый, 3

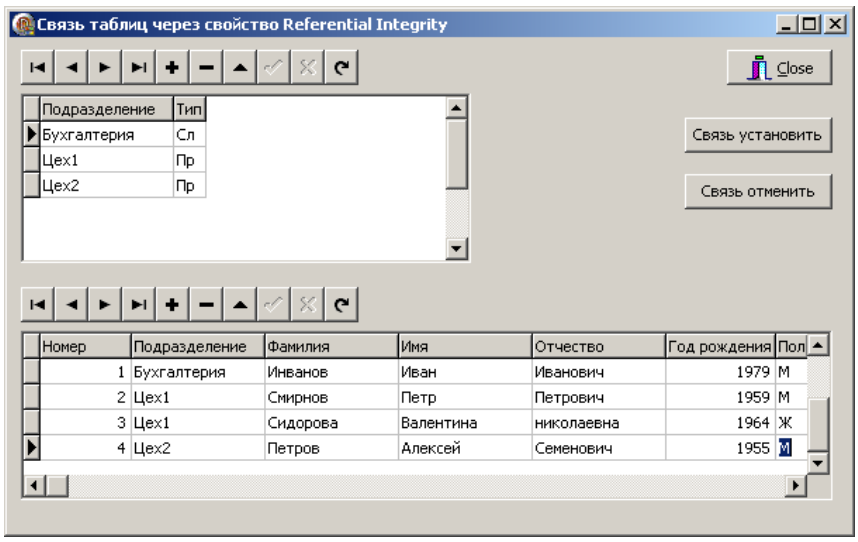
7. Для таблицы **Pers1** установите в качестве вторичных индексов поля **Dep2** (для связи типа **Referential Integrity** по этому полю индекс по **Dep2** обязателен), **Fam** и **God_R**. Для этого в окне **Create Paradox Table** из выпадающего списка с заголовком **Table Properties** выберите параметр **Secondary Indexes**. После выбора этого параметра из списка свойств появляются кнопки **Define (Определить)**, **Modify (Модифицировать)**, **Erase (Удалить)**. Для создания нового индекса нужно нажать кнопку **Define**. Появится окно **Define Secondary Index**, в нем задается состав полей и параметры индекса.



8. Перейдите в среду разработки приложений **Turbo Delphi**. Создайте новый проект, выполнив команду **File – New – VCL Forms Application**. Установите свойство формы **Caption** равным **Связь таблиц через свойство Referential Integrity**.
9. Нанесите на форму следующие компоненты для работы с базами данных: два компонента типа **Table**, два компонента типа **DataSource**, два компонента **DBGrid**, два компонента **DBNavigator**, кнопку типа **BitBtn** для закрытия приложения, для создания связи между таблицами две кнопки типа **Button**. Размещение компонентов на форме представлено на рисунке.

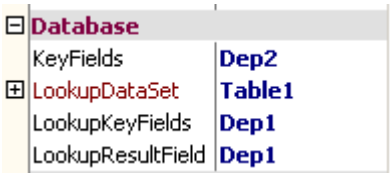


19. Установите для компонентов следующие свойства:
    - **Table1: DatabaseName=Dep2, TableName=Dep.db, IndexFieldNames=Dep1, Active=True,**
    - **Table2: DatabaseName=Dep2, TableName=Pers1.db, IndexFieldNames=Num, Active=True,**
    - **DataSource1: DataSet=Table1,**
    - **DataSource2: DataSet=Table2,**
    - **DBGrid1** и **DBNavigator1: DataSource= DataSource1,**
    - **DBGrid2** и **DBNavigator2: DataSource= DataSource2**
10. Установите русские заголовки столбцов в компонентах **DBGrid1** и **DBGrid2**. Заполните таблицы данными.



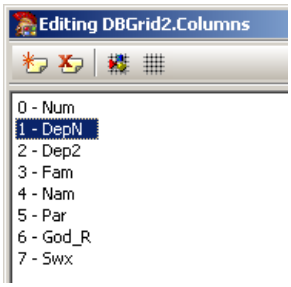
11. Для формирования связи между таблицами надо вызвать редактор полей таблицы **Table2**, добавить все поля. Вызвать его контекстное меню, выбрать в нем команду **New Field**. Появится окно **New Field**. В его разделе **Field Properties** установить свойства:
    - в окне **Name:** имя нового поля **DepN**;
    - в окне **Component**: появится имя нового поля в таблице **Table2DepN**;
    - в окне **Type**: выбрать из выпадающего списка требуемый тип нового поля **String**;
    - в окне **Size**: установить размер строки в символах **15.**
12. В разделе **Field Type** выбрать тип **Lookup** - выбор.
13. В разделе **Lookup Definition** (определение выбора) установить выбором из выпадающих списков значения:
    - в окне **Key Fields**: ключевое поле связи зависимой таблицы (**Table2**) - **Dep2**;
    - в окне **Data Set**: имя таблицы-источника допустимых значений - **Table1**;
    - в окне **Lookup Keys**: ключевое поле связи исходной таблицы (**Table1**) – **Dep1**;
    - в окне **Result Field**: имя поля из **Table1** для выпадающего списка в **Table2** – **Dep1**.
14. В результате формирования нового поля с выпадающим списком в редакторе таблицы **Table2** добавить все поля. Появиться и новое поле с его свойствами. Основные свойства поля с выпадающим списком:
    - **FieldKind = fkLookup**; - тип нового поля - выпадающее;
    - **FieldName = DepN**; - имя нового поля;
    - **KeyFields = Dep2**; - ключевое поле связи зависимой таблицы **Table2**;
    - **LookupDataSet =Table1**; - имя таблицы-источника допустимых значений;
    - **LookupKeyFields = Dep1**; - ключевое поле связи исходной таблицы - **Table1**;

- **LookupResultField = Dep1**; - имя поля из **Table1** для выпадающего списка.

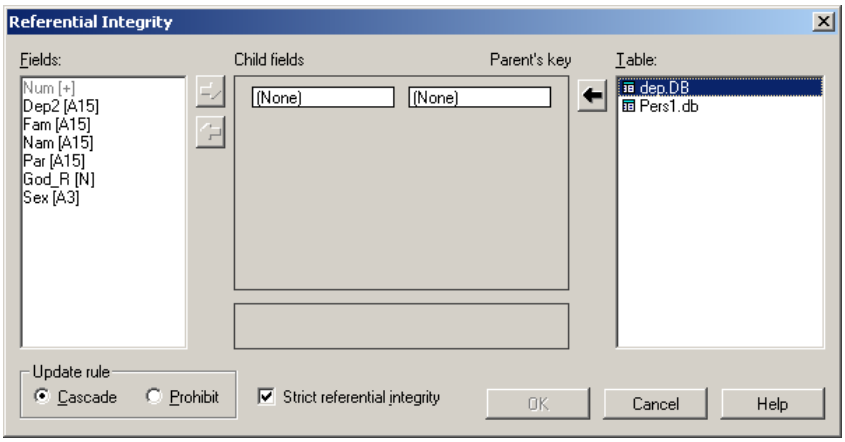


Поля связи не обязательно должны быть индексными. Поле связи (**Dep1**) не обязательно должно быть полем выпадающего списка (**Result Field = Dep1**).

15. Откройте окно редактора полей компонента **DBGrid2.** Выделите поле **Dep2** и установите его свойство **Visible** равным **False**. Выделите поле **DepN** установите его свойство **Title – Caption** равным **Подразделение** и поместите данное поле вторым в списке полей.

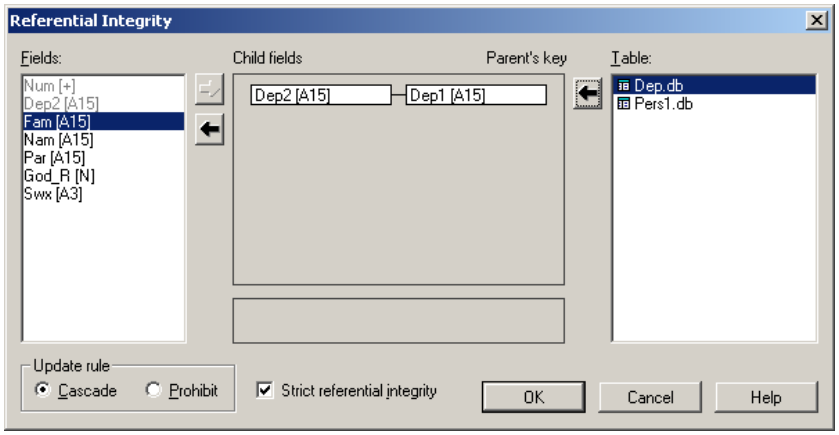


16. Для формирования связи подчиненной таблицы с основной с помощью свойства **Referential Integrity** необходимо запустить утилиту **Database Desktop** и открыть структуру таблицы **Pers1.db.**
17. В таблице свойств выберите свойство **Referential Integrity**. Нажмите кнопку **Define** (Определить связь). Появится форма **Referential Integrity**. В ней надо определить связь двух таблиц.

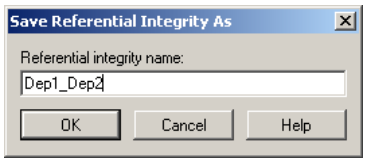


18. В окне **Fields**: формы **Referential Integrity** показан список полей записи подчиненной таблицы. Причем имена полей, которые не допускают создание связи, выделены серым

цветом. Выбрать поле **Dep2** для связи таблиц и нажать кнопку со стрелкой вправо. В поле **Child fields** появится имя выбранного поля и его тип (**Dep2[A12]**). В окне **Table**: список файлов каталога с базой данных. Надо выбрать главную таблицу для связи таблиц (**Dep.db**) и нажать кнопку со стрелкой влево. В поле **Parent's key** (родительские ключи) появится имя первого попавшегося подходящего поля для связи таблиц (**Dep1[A12]**).

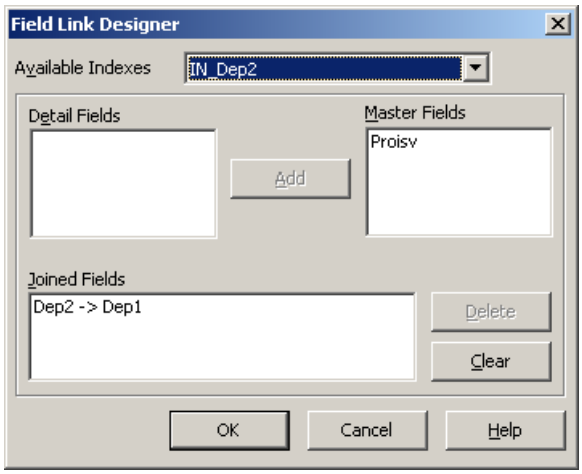


19. Группой переключателей **Update rule** можно установить, что будет, если в головной таблице удалить или изменить значение ключевого поля, с которым связаны записи подчиненной таблиц. Если установить опцию **Prohibit** (запретить), то такая операция будет недопустима. Если установить свойство **Cascade** (каскадное удаление записей), то:
    - при смене значений ключевого поля в головной таблице оно автоматически изменится в записях подчиненной таблицы;
    - если удалить запись в головной таблице, то в подчиненной таблице автоматически удалятся все записи, связанные с этим значением ключевого поля.
20. После определения связи **Referential Integrity** нажмите кнопку **ОК**. Появится окно для ввода имени созданной связи. Надо ввести в окно имя и нажать кнопку **ОК.**

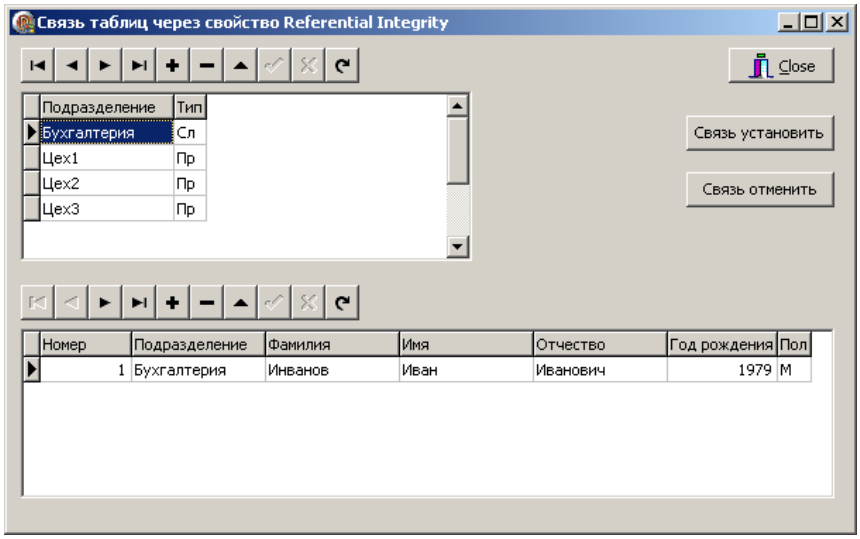


21. Для установки связи между таблицами в среде разработки приложения нужно для подчиненной таблицы (**Table2**) установить свойства:
    - **MasterSource = DataSource1;**
    - **IndexName = Ind_Dep2 (индекс поля связи).**

22. Затем нажать на многоточии в области значения свойства **MasterFields**. Появится окно **Field Link Designer** - для установки связи полей. Из выпадающего списка **AvailableIndexes** выбрать имя поля связи подчиненной таблицы **(Ind_Dep2).** В окне **Detail Fields** появится имя поля - **Dep2.** Выбрать в этом окне строку с **Dep2** и в окне **Master Fields** строку с именем **Dep1.** Активизируется кнопка **Add.** Нажать на кнопку **Add.** В окне **Joined Fields** (Связанные поля) появится строка с текстом: **Dep2-> Depl.** Кнопка **Add** снова станет недоступной.



23. В свойстве **MasterFields** подчиненной таблицы появится значение свойства, равное **Dep1.** Определение и установка связи типа **Referential Integrity** между таблицами завершены. После этого значение свойства **MasterFields = Dep1** можно устанавливать (для установки связи) вручную на этапе разработки приложения или программно в процессе выполнения приложения. Значение свойства **MasterFields**, равное пустой строке, разрывает эту связь.
24. На этапе разработки и выполнения приложения установка связи вызывает показ в подчиненной таблице только тех записей, у которых значение поля **Dep2** совпадает со значением поля **Dep1** головной таблицы. Вид формы при наличии связи таблиц показан на рисунке **(MasterFields = Dep1).**



25. При отсутствии связи на этапе разработки и выполнения приложения можно сортировать данные с помощью свойств **IndexName** или **IndexFieldName**. После этого надо убрать значения названных свойств, чтобы снова установить связь таблиц с помощью свойства **Table2.MasterFields:**
    - для **Table2.MasterFields := 'Depl'** - связь устанавливается;
    - для **Table2.MasterFields :=** '' - связь отменена.
26. Выполните двойной щелчок по кнопке **Связь установить** и в заготовке процедуры запишите следующий код:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
Table2.MasterFields := 'Dep1';
end;
```

27. Выполните двойной щелчок по кнопке **Связь отменить** и в заготовке процедуры запишите следующий код:

```
procedure TForm2.Button2Click(Sender: TObject);
begin
Table2.MasterFields := '';
end;
```

28. Протестируйте работу приложения. Сохраните изменения.

Внеаудиторная самостоятельная работа:

Ответьте на вопросы письменно в тетради:

- Как установить связь полей основной и зависимой таблиц при разработке структуры записи зависимой таблицы?
- Как установить текст выводимых значений в поле булевского типа компонента DBGrid?
- Как заполнить выпадающий список зависимой таблицы допустимыми значениями?
- Как сформировать новое поле для выбора допустимых значений поля из выпадающего списка допустимых значений?
- Как сделать невидимым одно из двух идентичных полей таблицы?
- Какую зависимость таблиц может реализовать свойство Referential Integrity?
- Как определить связь таблиц типа Referential Integrity с помощью Database Desktop?
- Как установить (отменить) связь двух таблиц на этапе разработки приложения?
- Как установить (отменить) связь двух таблиц на этапе выполнения приложения?

Практическая работа №10

Разработка приложения для базы данных MS Access в Delphi. Основные свойства и методы компонента ADOConnection.

Цель работы: изучить основные возможности механизма доступа к данным **Microsoft AvtiveX Data Objects (ADO)**; изучить основные возможности компонентов **TADOConnection** и **TADOTable**; сформировать умения по проектированию базы данных средствами **MS Access 2007**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **MS Access 2007**.

Теоретический материал:

Механизм доступа к данным - это программный инструмент, позволяющий получить доступ к базе данных и ее таблицам. Как правило, это драйвер в виде ***.dll** файлов, который устанавливается на ПК разработчика (и клиента), и который используется программой для связи с БД.

Borland Database Engine (BDE) - данный механизм доступа к данным позволяет обращаться к локальным и файл-серверным форматам баз данных dBase, FoxPro и Paradox, к различным серверам SQL и ко многим другим источникам данных, доступ которых поддерживался при помощи драйверов ODBC. Механизм доступа BDE признается устаревшим даже самой компанией Borland. В данный момент многие инструменты Delphi являются кросс - платформенными, то есть, программы с небольшими доработками можно переносить на другие операционные системы. Корпорация Borland выпустила новую среду быстрой разработки программ - **Kylix**, на которой создаются приложения для операционных систем семейства Linux. Часто говорят, что Kylix - это Delphi для Linux. Большинство инструментов Delphi были унаследованы Kylix, но, увы, не BDE. Дальнейшее развитие этого механизма доступа к данным корпорацией Borland прекращено.

Тем не менее, многие программисты до сих пор используют данный инструмент в разработке приложений для небольших компаний. Например, китайская компания **Huawei**, разрабатывающая современнейшие электронные АТС как для городских, так и для мобильных телефонов, до сих пор использует BDE для доступа к настройкам и статистическим данным этих АТС. Кроме того, BDE имеет множество простых и удобных возможностей для программиста, таких например, как создание таблиц программно.

Удобство работы с BDE трудно переоценить, однако нельзя не сказать и о минусах. Основной минус - распространение приложений. Если ваше приложение использует для доступа к данным компоненты BDE, то и у клиента, который будет пользоваться вашей программой, должен быть установлен BDE. Причем если вы использовали алиасы (псевдонимы базы данных), то настройка на эти же алиасы должна быть и у клиента.

Другой минус касается не только BDE, но и любого другого универсального механизма доступа к данным. Универсальность такого механизма подразумевает сложность его реализации. Программисту предоставляется уже готовый инструмент, с которым удобно работать, однако этот инструмент достаточно «тяжелый» - используя его, вы довольно существенно увеличиваете размеры своего приложения.

ActiveX Data Object (ADO) - это механизм доступа к данным, разработанный корпорацией Microsoft. Если точнее, то ADO - это надстройка над технологией OLE DB, посредством которой можно связываться с различными данными приложений Microsoft. В середине 1990-х годов большое развитие получила технология COM, и корпорация Microsoft в связи с этим объявила о постепенном переходе от старой технологии ODBC к новой OLE DB. Однако технология OLE DB достаточно сложная, использование этой технологии происходит на системном уровне и требует от программиста немало знаний и труда. Кроме того, технология OLE DB очень чувствительна к ошибкам, и «вылетает» при первом удобном случае. Чтобы облегчить программистам жизнь, корпорация Microsoft разработала дополнительный прикладной уровень ADO, который мы будем изучать на этом курсе.

По своим возможностям ADO напоминает BDE, хотя конечно, является более мощным инструментом. Компания Borland разработала набор компонентов для доступа к ADO и первоначально назвала его **ADOExpress**. Однако корпорация Microsoft упорно противится использованию своих обозначений в продуктах сторонних разработчиков, поэтому, начиная с Delphi 6, этот набор компонентов стал именоваться **dbGo**. Эти компоненты вы можете увидеть на вкладке ADO палитры компонентов.

Технология ADO, как и BDE, независима от конкретного сервера БД, имеет поддержку как локальных баз данных различных типов, так и некоторых клиент-серверных БД. Плюсов у этой технологии много. Драйверы, разработанные корпорацией Microsoft для собственных нужд, более надежные, чем драйверы сторонних производителей. Поэтому если вам требуется работать с базами данных MS Access или для архитектуры клиент-сервер использовать MS SQL Server, то использование ADO будет наиболее предпочтительным. Кроме того, имеется плюс и в вопросе распространения программ - во всех современных Windows встроены драйверы ADO. Другими словами, ваша программа будет работать на любом ПК, где установлен Windows.

Основной минус так же заключается в вопросе распространения программ. Корпорация Microsoft поступает довольно хитро. Каждые пару-тройку лет появляются новые версии Windows. Рядовому пользователю обычно нет нужды переходить на свежую ОС, тем более что каждая новая система становится все требовательней к ресурсам ПК. Для того чтобы заставить пользователя перейти на новую версию, корпорация Microsoft обязательно вводит несколько новых стандартов или технологий, несовместимых со старыми.

Технология ADO на самом деле является частью Microsoft Data Access Components (MDAC). Компания Microsoft распространяет MDAC как отдельный продукт, к счастью, бесплатный. При этом поддерживается только самая последняя версия MDAC. Например, в состав Delphi 7 входит MDAC 2.6. При распространении собственных программ следует учитывать, что у клиента с большей долей вероятности уже установлена эта самая MDAC, причем самой последней версии.

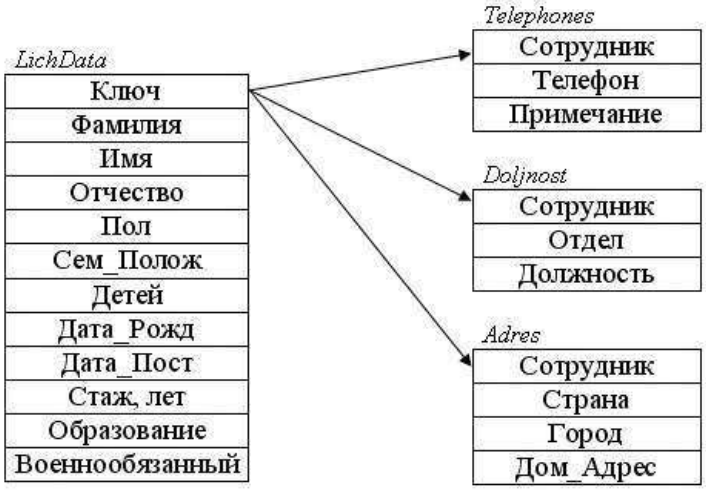
Однако если он пользуется старыми версиями Windows (Win95, 98, ME, NT), то вам потребуется позаботиться об установке MDAC на его компьютер. Если же у него установлена ОС Win2000, WinXP или более новая, то MDAC у него уже есть, и вам беспокоиться не о чем.

Еще один серьезный минус ADO в том, что он для подключения к БД использует довольно медлительную технологию COM. Если ваша база данных будет содержать несколько тысяч записей, то скорость работы с таблицами может стать в сотни раз более медленной, чем если бы вы использовали BDE. На современных ПК, имеющих частоту процессора до 2 ГГц и выше, эти замедления могут быть и незаметны, но работа с огромной базой данных на более медленных ПК превратится в сплошное ожидание.

Задание 1. Создание базы данных в MS Access

Методические указания по выполнению задания

1. Создадим базу данных для отдела кадров какого-нибудь предприятия. Оптимизируем данные, исходя из правил трех нормальных форм. В результате получаем следующие четыре таблицы:



Главной в данном случае будет таблица **LichData**, которая содержит основные данные о сотруднике. Она имеет релятивные связи с другими таблицами. Поле **«Ключ»** будет автоинкрементным, то есть автоматически будет прибавляться на единицу, гарантируя нам уникальность ключа. В подчиненных таблицах имеется поле **«Сотрудник»** целого типа, по которому будет обеспечиваться связь. Причем ключевых полей в дочерних таблицах не будет.

Главная таблица поддерживает связь один-к-одному с таблицами **Doljnost** и **Adres**, и связь один-ко-многим с таблицей **Telephones**, ведь у сотрудника наверняка есть и домашний, и рабочий телефоны, то есть, один сотрудник может иметь много телефонов.

2. Запустите программу **MS Access**. Щелкните по кнопке **Office** и выберите команду **Создать**. В правой части окна появиться панель **Новая база данных.** Далее необходимо указать имя нашей базы данных, для этого создайте папку **Base_1** в папке **Мои документы – 3 АСОИ**. Укажите в

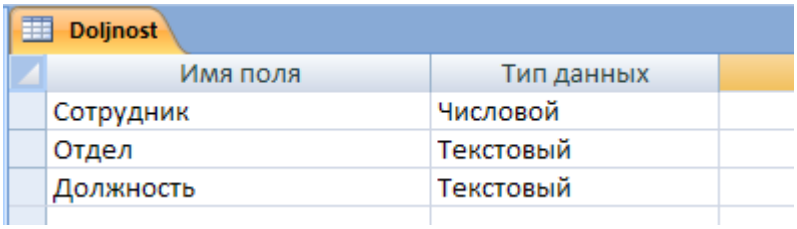
качестве места сохранения эту папку и задайте имя базы данных **ok.mdb** (в поле тип файла выберите **Базы данных Microsoft Office Access 2002-2003 (*.mdb)**). Щелкните по кнопке **Создать**. После этого появиться окно базы данных с новой таблицей запущенной в режиме таблицы.

3. Сейчас нам потребуется сделать четыре таблицы. Поэтому переходим в режим конструктора таблиц. Задаем имя таблицы **LichData**. В левой части окна конструктора мы вводим имя поля, причем русскими буквами. В поле **Тип данных** выбираем тип, а на вкладке **Общие** делаем настройки поля. Описание поля заполнять необязательно. Итак, создаем поля:

Имя поля	**Тип данных**	**Дополнительные параметры**
Ключ	Счетчик	ключевое поле
Фамилия	Текстовый	размер – 25 символов, индексированное поле – Да (Допускаются совпадения)
Имя	Текстовый	размер – 25 символов, индексированное поле – Да (Допускаются совпадения)
Отчество	Текстовый	размер – 25 символов
Пол	Текстовый	размер 3 символа, формат поля – муж/жен
Сем_Полож	Логический	формат поля – Да/Нет
Детей	Числовой	размер поля – Байт
Дата_Рожд	Дата/Время	формат – Краткий формат даты, маска ввода - 00.00.0000
Дата_Пост	Дата/Время	формат – Краткий формат даты, маска ввода - 00.00.0000
Стаж	Числовой	формат поля - Байт
Образование	Текстовый	размер – 30 символов
Военнообязанный	Логический	формат поля – Да/Нет

4. Закрываем таблицу, на запрос о сохранении таблицы отвечаем утвердительно. Главная таблица нами создана.
5. Переходим на вкладку **Создание** и выбираем команду **Конструктор таблиц**. Создаем следующие поля таблицы:

Имя поля	**Тип данных**	**Дополнительные параметры**
Сотрудник	Числовой	размер поля – длинное целое
Отдел	Текстовый	размер – 15 символов
Должность	Текстовый	размер – 20 символов



6. Закрываем таблицу, дав ей имя **Doljnost**. На запрос о создании ключевого поля отвечаем отказом.
7. Переходим на вкладку **Создание** и выбираем команду **Конструктор таблиц**. Создаем следующие поля таблицы:

Имя поля	**Тип данных**	**Дополнительные параметры**
Сотрудник	Числовой	размер поля – длинное целое
Страна	Текстовый	размер – 15 символов
Город	Текстовый	размер – 20 символов
Дом_Адрес	Текстовый	размер – 100 символов

8. Закрываем таблицу, дав ей имя **Adres**. На запрос о создании ключевого поля отвечаем отказом.
9. Переходим на вкладку **Создание** и выбираем команду **Конструктор таблиц**. Создаем следующие поля таблицы:

Имя поля	**Тип данных**	**Дополнительные параметры**
Сотрудник	Числовой	размер поля – длинное целое
Телефон	Текстовый	размер – 17 символов. Желательно задать маску. После выбора данного свойства выйдет запрос о сохранении таблицы, сохраните ее под именем **Telephones**, ключевые поля создавать не нужно. Далее в окне нажимаем кнопку **Список**. Настраиваем маску: **Описание** – Телефон, **Маска ввода** - #(###)-###-##-##
Примечание	Текстовый	размер – 10 символов, формат поля - Рабочий/Домашний/Мобильный

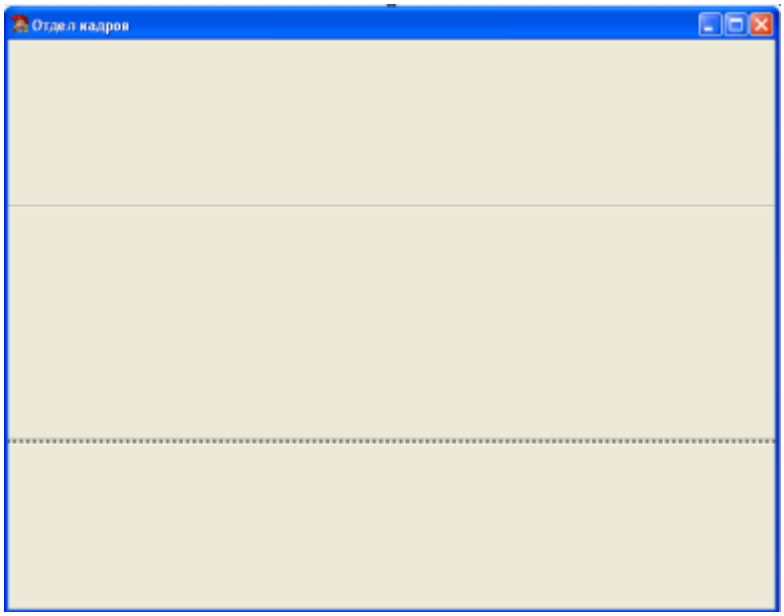
10. Закрываем таблицу, на запрос о сохранении таблицы отвечаем утвердительно.
11. База данных готова. Программу **MS Access** можно закрыть, больше она не понадобится. Пока база данных еще пустая, желательно сделать резервную копию файла **ok.mdb**, который и является полученной базой данных. Как видите, никаких связей между таблицами мы не делали - проще будет сделать их в проекте программы.

Задание 2. Размещение и настройка основных компонентов главной формы приложения

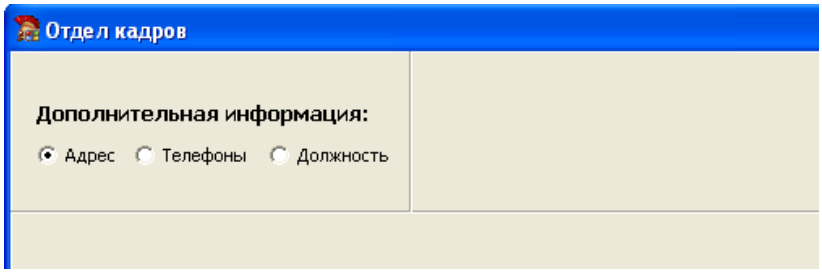
Методические указания по выполнению задания:

1. Запустите интегрированную среду разработчика **Turbo Delphi**.
2. Создайте новый проект **Delphi**. Сохраните проект в каталоге **Base_1,** выполнив команду **File – Save Project as,** файл модуля сохраните под именем под именем **Main.pas**, а файл проекта под именем **OK.dpr.**
3. Установите свойство формы **Caption** равным **«Отдел кадров»,** а свойство **Name** равным **fMain**.

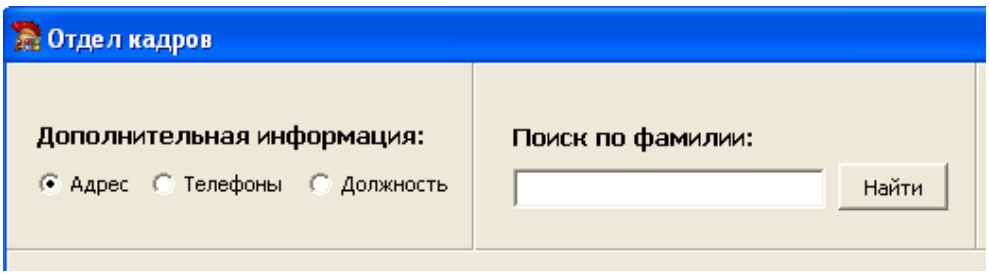
4. Разместите на форме три компонента **TPanel**. Свойству **Align** верхней панели присвойте значение **alTop**. Затем свойству **Align** нижней панели присвойте значение **alBottom**. Затем поместите компонент **Splitter** с вкладки **Additional** панели инструментов, и его свойству **Align** также присвойте значение **alBottom**, после этого он прижмется к нижней панели. **Splitter** - это разделитель между панелями. С его помощью пользователь мышью сможет передвигать нижнюю панель, меняя ее размеры. Свойству **Align** средней панели присвойте значение **alClient**, чтобы она заняла все оставшееся место на форме. Не забудьте очистить свойство **Caption** всех трех панелей.



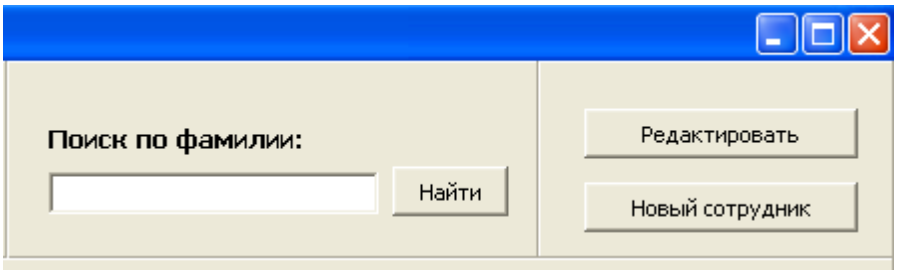
5. На верхней панели поместите три компонента **RadioButton** с вкладки **Standard** палитры компонентов. В их свойствах **Caption** напишите, соответственно, **Адрес**, **Телефоны** и **Должность**. Переключаясь между ними, пользователь сможет выводить в нижнюю, подчиненную сетку **DBGrid** нужные данные. Свойству **Checked** первой радиокнопки присвойте значение **True**, чтобы включить ее. Раздел с переключателями разделите компонентом **Bevel** с вкладки **Additional** палитры компонентов. Его ширину (свойство **Width**) задайте равным 2 пикселям, превратив его в вертикальную разделительную полосу.



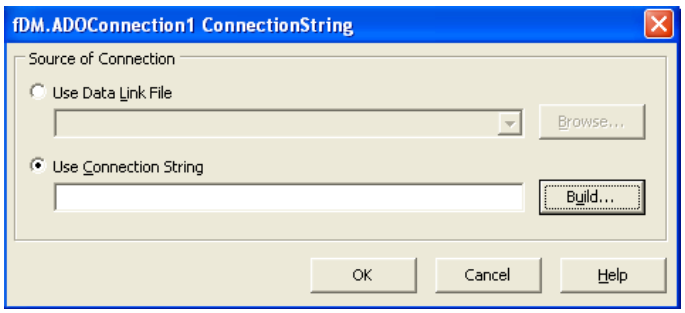
6. Далее создадим раздел поиска, поместив в него обычные **Label**, **Edit** и кнопку **BitBtn**. Задайте свойства компонентов в соответствии с рисунком.



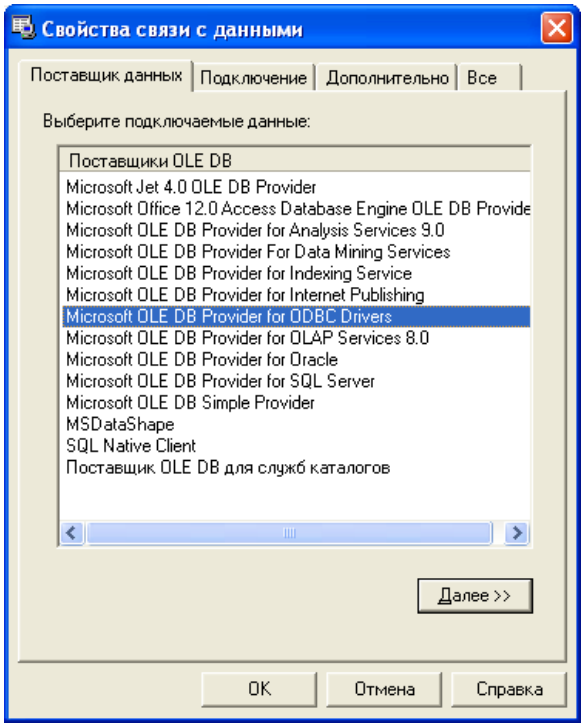
7. В последнем разделе верхней панели находятся еще две кнопки **BitBtn**. Одна из них предназначена для редактирования текущей записи, другая - для добавления новой.



8. Вторая и третья панели содержат только по одному компоненту **DBGrid** с вкладки **DataControls** палитры компонентов, свойствам **Align** которых присвоено значение **alClient**.
9. Далее добавим в проект модуль данных с помощь команды **File - New – Other,** а в окне **New Items** выберем **Data Module**. Модуль данных - это не визуальный контейнер для размещения на нем не визуальных компонентов. Он предназначен для размещения в нем компонентов подключения к данным (**TDataBase, ADOConnection** и т.п.), компонентов - наборов данных (**TTable/ADOTable, TQuery/ADOQuery, TStoredProc/ADOStoredProc**) и компонентов **DataSource**, которые обеспечивают связь наборов данных и компонентов отображения/редактирования данных. Также модуль данных часто используют и для хранения глобальных переменных, общих функций и процедур, которые должны быть видны по всей программе. Модуль данных не имеет формы, но сохраняется как модуль в файле ***.pas**.
10. Свойству **Name** модуля данных присвоим имя **fDM**, а модуль сохраним как **DM.pas**. Далее добавляем в модуль компонент **ADOConnection** с вкладки **dbGo** палитры компонентов. Этот компонент обеспечит связь других компонентов с базой данных при помощи механизма **ADO**.
11. Связь обеспечивается свойством компонента **ConnectionString**. Щелкните дважды по свойству **ConnectionString** компонента **ADOConnection**. Откроется окно подключения компонента к **ADO**.



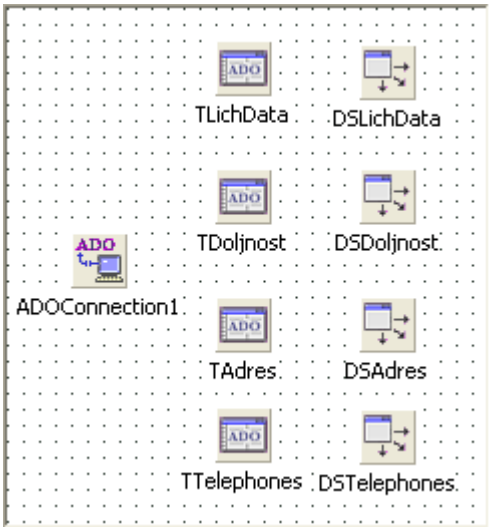
12. В данном окне имеется возможность подключиться тремя способами: использовать для связи созданный ранее link-файл; вписать в поле **Use Connection String** строку для связи с ADO; сгенерировать эту строку, нажав кнопку **Build**. Воспользуемся третьим способом - нажмем кнопку **Build**. Открывается новое окно, содержащее настройки подключения.



13. Вначале нам предлагается выбрать поставщика OLE DB, или иначе, указать нужный для подключения драйвер. Для связи с базой данных MS Access больше всего подходит **Microsoft Jet 4.0 OLE DB Provider. Jet** - это название механизма работы с СУБД, встроенного в MS Access. Этот механизм поддерживает как собственные БД MS Access, имеющие расширение *.mdb, так и ODBC. Его и выделяем в списке.
14. Нажимаем на кнопку **Далее**, либо переходим к вкладке **Подключение**. Здесь нам нужно выбрать или ввести базу данных. Если мы выберем базу данных, то есть, нажмем на кнопку с тремя точками, откроем диалог выбора и найдем там наш файл, то база данных будет привязана к указанному адресу. Если вы желаете поместить базу данных в какой-то определенной папке, то так и поступите. Однако если вы поместили файл с базой данных (в нашем случае ok.mdb) там же, где находится программа, и не желаете зависеть от определенной папки (ведь

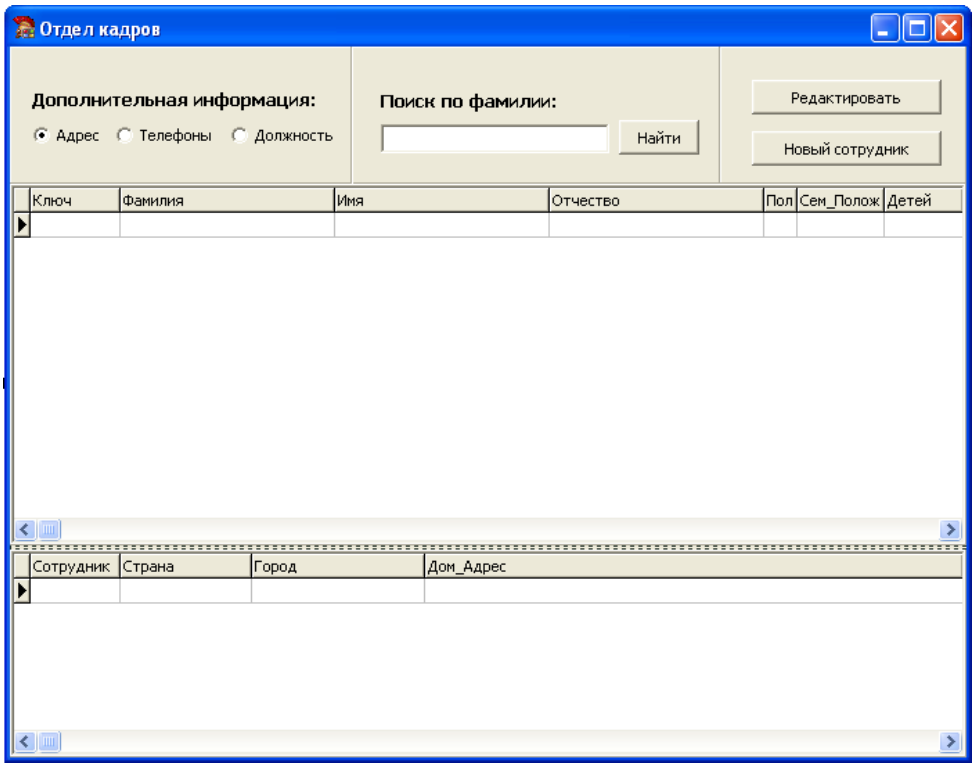
пользователь может переместить вашу программу), то нужно вручную вписать только имя файла с БД, без всякого адреса. В этом случае вы не сможете проверить подключение, нажав на кнопку **Проверить подключение**. Укажите только имя файла - **ok.mdb**. Нажмите на кнопку **ОК**.

15. Закрываем окно редактора связей, и нам остается открыть подключение. Однако перед этим переведите свойство **LoginPrompt** компонента **ADOConnection** в **Fals**e. Если этого не сделать, то при каждой попытке соединиться с базой данных будет выходить запрос на пользовательское имя и пароль, нам это не нужно, наша база данных без пароля. Теперь свойство **Connected** переведите в **True**. Если вам удалось это сделать, и не вышло никаких сообщений об ошибке, то подключение состоялось.
16. Установите в модуль данных четыре компонента **ADOTable**, по одному на каждую таблицу из нашей базы данных. Компонент **ADOTable** (также как и **TTable**) предназначен для создания набора данных.
17. Выделите все четыре компонента **ADOTable** (удерживая клавишу **Shift**), и в их свойстве **Connection** выберите нашу связь **ADOConnection1**. Таким образом, все четыре **ADOTable** мы подключили к базе данных.
18. Выделите первый компонент **ADOTable**. Переименуйте его свойство **Name** в **TLichData**, а в свойстве **TableName** выберите главную таблицу базы - **LichData**. Рядом с компонентом установите компонент **DataSource** из вкладки **Data Access** палитры компонентов. Свойство **Name** компонента **DataSource** переименуйте в **DSLichData**. В свойстве **DataSet** данного компонента выберите таблицу **TLichData**.
19. То же самое нужно проделать еще три раза, подключая аналогичным образом компоненты **DataSource** к другим таблицам.



20. Свойство **Active** таблиц переведите в **True**, открыв их.

21. Перейдите на главную форму. Выберите команду **File - Use Unit** и подключите к ней модуль **DM**. Теперь мы сможем видеть таблицы из главной формы.
22. Выделите верхнюю сетку **DBGrid**, в ее свойстве **DataSource** выберите **fDM.DSLichData**. В таком же свойстве нижней сетки выберите **fDM.DSAdres**. Сетки среагировали, и вы можете видеть названия полей. Разумеется, таблица еще пуста, данных пока нет. Кстати, выделите обе сетки, и установите в **True** их свойства **ReadOnly** - только чтение. Таблицы ведь будут связаны, и нам не нужно, чтобы пользователь вводил данные фрагментарно. Мы для этого сделаем отдельную форму, а эти сетки нужны только для просмотра.

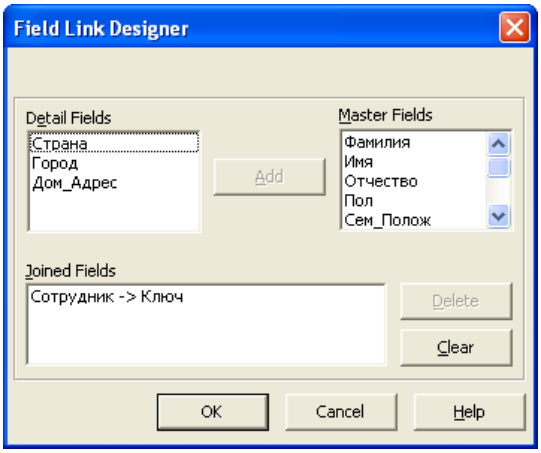


23. Теперь нужно между таблицами установить связь. Это требуется не только для того, чтобы в нижней сетке выходили данные только на сотрудника, выделенного в верхней сетке, но и для того, чтобы мы смогли в дальнейшем вводить связанные данные в окне редактора. Снова перейдите в модуль данных. Щелкните дважды по первой таблице, чтобы открыть редактор полей. Правой кнопкой щелкните по этому редактору и выберите команду **Add all fields** (добавить все поля). В окне редактора полей появились все поля таблицы.

    Редактор полей предназначен для настройки параметров каждого поля, для добавления новых полей или удаления имеющихся. Если в редакторе полей нет ни одного поля, то в компоненте **DBGrid** будут отображены все поля таблицы, имеющие параметры по умолчанию. Если же мы добавили в редактор полей хотя бы одно поле, то сетка **DBGrid** его и отобразит. В редакторе мы можем для каждого поля изменить различные параметры, например, ширину колонки, название колонки, видимое это поле или нет, и т.п. Кроме того, редактор полей предоставляет

возможность добавлять в набор данных новые поля, например вычисляемые или просматриваемые (lookup).

24. Поле **Ключ** у нас автоинкрементное, предназначено для связи с другими таблицами. Пользователю его видеть не обязательно. Выделите его, и в свойстве **Visible** установите **False.** Теперь для пользователя оно будет невидимым. У нас есть два логических поля - **Сем_Полож** и **Военнообязанный**. Чтобы **True** и **False** выходили на экране так, как нам нужно, свойству **DisplayValues** первого из этих полей присвойте значение **Женат;Холост**, а второго - **Да;Нет.** Первым здесь идет значение, которое будет обозначать **True**, вторым - **False**. Эти значения разделяются точкой с запятой, пробелы не нужны.
25. Таким же образом добавьте все поля в остальные три таблицы. У них невидимым следует сделать поле **Сотрудник** - этому полю автоматически будет присвоено такое же число, как у поля **Ключ** соответствующей записи. Логических полей у них нет. Однако для поля **Телефон** таблицы **Telephones** следует изменить свойство **EditMask**. Щелкните по нему дважды, открыв редактор маски, и в поле **Input Mask** введите маску **#(###)-###-##-##**. Сохраните ее, нажав кнопку **ОК**. Для полей типа **Дата** в этом свойстве (в таблице **LichData** два таких поля) введите маску **##.##.####.**
26. Для установки связи между таблицами в среде разработки приложения нужно для подчиненной таблицы **TAdres** установить свойство **MasterSource = DSLichData**. Затем нажать на многоточии в области значения свойства **MasterFields**. Появится окно **Field Link Designer** - для установки связи полей. В окне **Detail Fields** выбрать строку **Сотрудник** и в окне **Master Fields** строку с именем **Ключ**. Активизируется кнопка **Add**. Нажать на кнопку **Add**. В окне **Joined Fields** появится строка с текстом: **Сотрудник-> Ключ**. Кнопка **Add** снова станет недоступной. Нажать кнопку **ОК.** В свойстве **MasterFields** подчиненной таблицы появится значение свойства, равное **Ключ**. Установите связи между оставшимися таблицами, выполнив аналогичные действия.

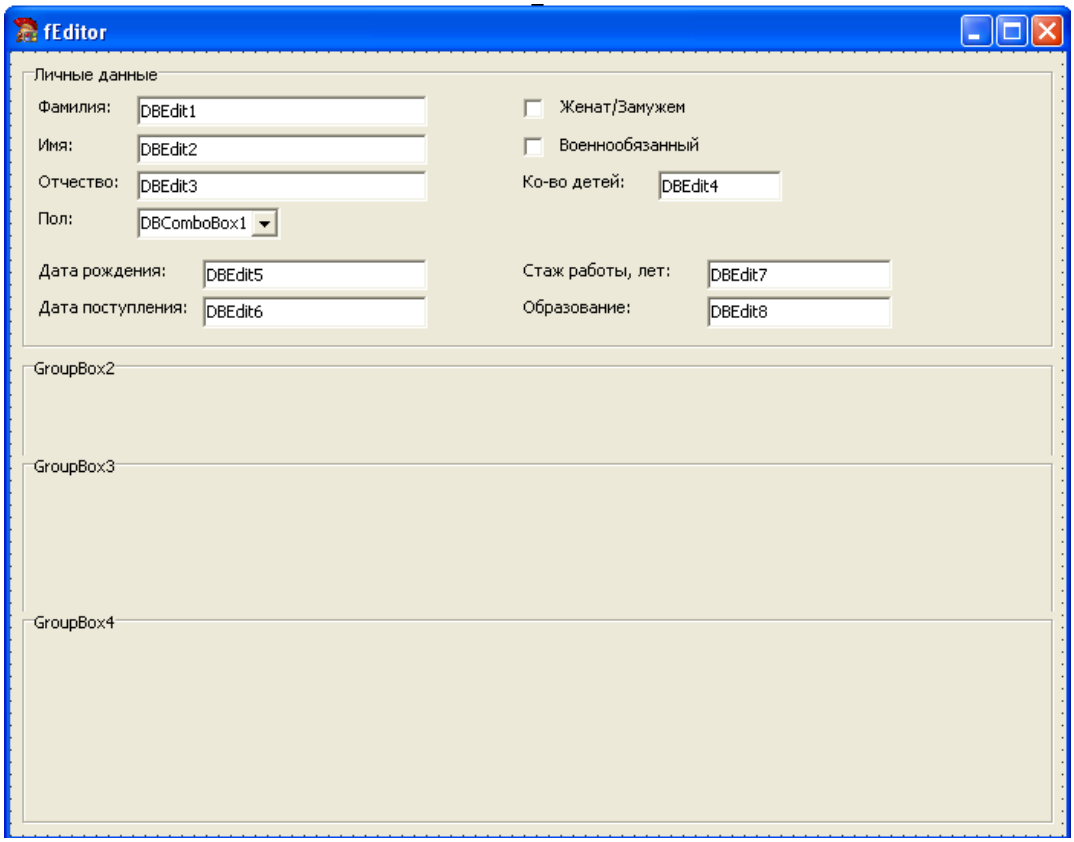


27. Сохраните проект, скомпилируйте его и запустите на выполнение. Если в сетках главного окна вы видите открытые таблицы, то все хорошо. Если нет, возможно, при изменении настроек ваши таблицы закрылись. В таком случае закройте программу, выделите таблицы, и их свойству **Active** снова присвойте значение **True**. Таблицы должны появиться в сетках главного окна, даже на этапе проектирования.

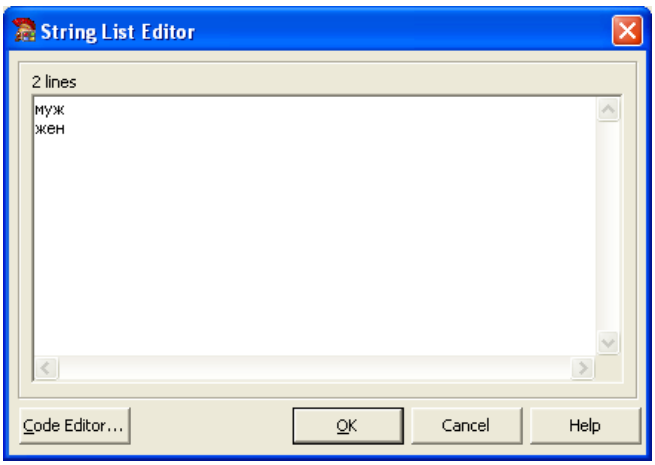
Задание 3. Разработка окна редактора данных

Методические указания по выполнению задания:

1. Создайте новую форму, выполнив команду **File - New - Form**.
2. Свойство **Name** данной формы переименуйте в **fEditor**, а при сохранении формы дайте модулю имя **Editor**.
3. Командой **File - Use Unit** подключите к форме модуль данных **DM**.
4. Далее нам нужно разместить на форме основные компоненты. Установите на форме четыре панели **GroupBox** с вкладки **Standard**, для каждой таблицы свой **GroupBox.**
5. Займемся первой таблицей. В свойстве **Caption** компонента **GroupBox** напишите **Личные данные**, это название отразится в заголовке панели. Далее на эту панель следует установить восемь компонентов **DBEdit** с вкладки **DataControls** палитры компонентов, два **DBCheckBox** для редактирования логических данных, и один компонент **DBComboBox** для списка. Поясняющие компоненты **Label** установите и настройте самостоятельно.

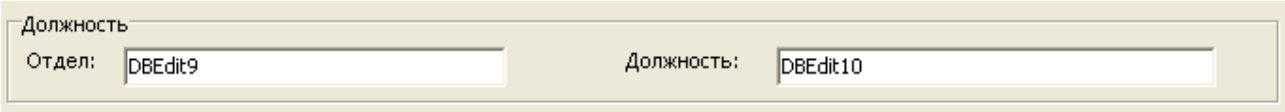


6. Доработаем компонент **DBComboBox**, для этого щелкните дважды по его свойству **Items**, открыв редактор. В нем введите две строки:

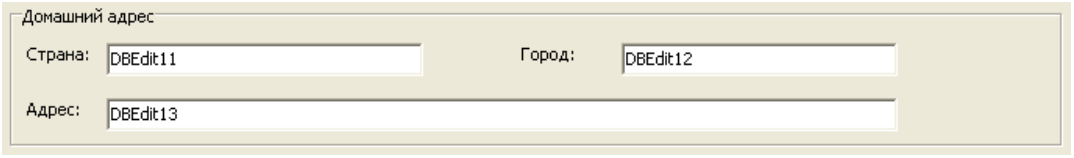


Сохраните текст, нажав кнопку **ОК**. Теперь пользователь сможет указать пол сотрудника, выбрав нужную строку из списка.

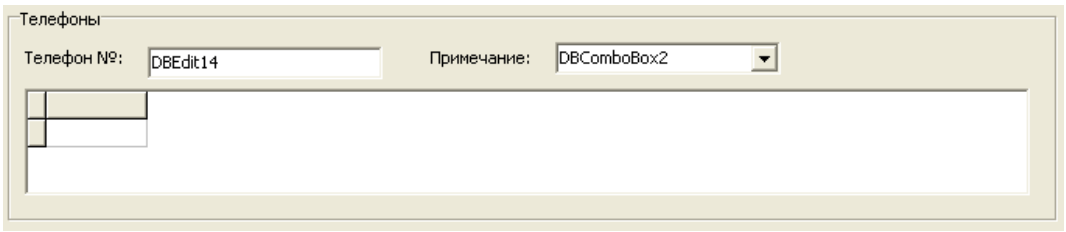
7. Для таблицы **Doljnost** разместите на панели **GroupBox** всего два компонента **DBEdit** и два поясняющих **Label**.



8. Для таблицы **Adres** используйте три **DBEdit** и соответствующие поясняющие надписи.



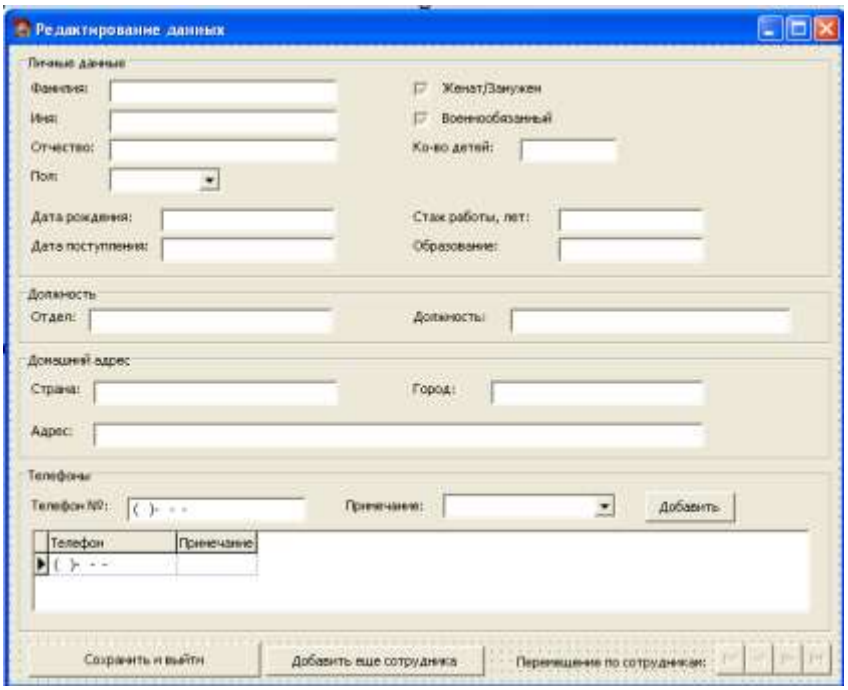
9. Для таблицы **Telephones** понадобится один **DBEdit**, один **DBComboBox**, сетка **DBGrid** и кнопка **BitBtn**. Сетка нужна для контроля введенных телефонов, ведь здесь связь один-ко-многим, и телефонов может быть несколько. В редакторе **Items** компонента **DBComboBox** введите три строки: **Рабочий, Домашний, Мобильный.**



10. Теперь займемся подключением компонентов контроля. Удерживая клавишу **Shift**, выделите все компоненты контроля на первой панели (все компоненты, кроме **Label**). Их свойство **DataSource** установите равным **fDM.DSLichData**, подключив компоненты к нужному набору данных (таблице). Снимите общее выделение, и выделите первый компонент **DBEdit**. В его свойстве **DataField** выберите поле **Фамилия**. Это свойство подключает выбранный компонент

к определенному полю таблицы. Таким же образом подключите к соответствующим полям остальные компоненты.

11. Затем подключайте компоненты других таблиц, каждое к своей таблице и к соответствующему полю. Сетка **DBGrid** подключается к **fDM.DSTelephones**, и не имеет поля, разумеется. Она отображает все видимые поля таблицы.



12. В правой нижней части для удобства пользователя установите навигационный компонент **DBNavigator** с вкладки **Data Controls**. Этот компонент предназначен для перемещения по записям, включения режима редактирования записи, сохранения или отмены сделанных изменений, добавления новой записи или удаления существующей. В его свойстве **DataSource** выберите **fDM.DSLichData**, чтобы подключить компонент к главной таблице. Нам нужна от этого компонента только возможность перехода на начало или конец таблицы, на следующую или предыдущую запись. Поэтому раскройте его свойство **VisibleButtons** и переведите в **False** все кнопки, кроме **nbFirst**, **nbPrior**, **nbNext** и **nbLast**. Нажатие на эти кнопки приведет к вызову соответствующих методов компонента **ADOTable**. Эти методы делают следующее: **First** - переход на первую запись таблицы; **Prior** - переход на предыдущую запись; **Next** - переход на следующую запись; **Last** - переход на последнюю запись.
13. Когда у **DBNavigator** останется всего четыре кнопки, эти кнопки окажутся вытянутыми. Уменьшите ширину компонента, чтобы кнопки приняли более привычный вид.
14. Компоненты нами были помещены на панели **GroupBox**, поскольку, как вы уже знаете, измененная запись в таблице сохраняется в трех случаях: применением метода Post; при переходе на другую запись; при добавлении новой записи. Когда мы, заполнив одну таблицу, перейдем к другой, то в первой таблице запись еще не будет сохранена. Поле **Ключ** у нас

автоинкрементное, на него завязаны остальные таблицы. До тех пор, пока мы не сохраним запись, в этом поле не будет никакого значения. Следовательно, данные в других таблицах не смогут привязаться к какой-то записи главной таблицы. Поэтому выделите первый **GroupBox**, и дважды щелкните по событию **onExit** на вкладке **Events** инспектора объектов. Это событие происходит всякий раз, когда пользователь перейдет к другой панели **GroupBox**, либо к кнопкам, расположенным в нижней части окна. В сгенерированной процедуре впишите код:

```
procedure TfEditor.GroupBox1Exit(Sender: TObject);
begin
 if fDM.TLichData.Modified then  fDM.TLichData.Post;
end;
```

Свойство **Modified** компонента **ADOTable** имеет логический тип - в нем содержится **True**, если данные были изменены, и **False** в противном случае. Метод **Post** этого компонента, как уже упоминалось, сохраняет измененную запись таблицы. При этом в поле **Ключ** попадет присвоенное автоматически значение. Таким образом, введенный код означает, что если запись была изменена, то следует ее сохранить.

15. Сгенерируйте событие **onExit** для оставшихся панелей **GroupBox** и таким же образом сохраните изменения записей в соответствующих таблицах.
16. Далее сгенерируйте событие нажатия на кнопку **Добавить** в **GroupBox** с телефонными данными. Этой кнопкой мы будем добавлять новые записи в таблицу, ведь один сотрудник может иметь более одного телефона. Код в процедуре будет такой:

```
procedure TfEditor.Button2Click(Sender: TObject);
begin
if fDM.TTelephones.Modified then fDM.TTelephones.Post;
  fDM.TTelephones.Append;
  DBEdit14.SetFocus;
end;
```

Вначале мы сохраняем измененные значения, если они были. Затем методом **Append** мы добавляем в таблицу новую запись. Добавить новую запись можно двумя методами: **Append** - добавляет новую запись в конец таблицы, **Insert** - добавляет новую запись в текущее положение курсора.

После добавления новой записи таблица уже будет в режиме редактирования, поэтому можно не вызывать метод **Edit**, который переводит таблицу в этот режим. Далее мы переводит фокус ввода на **DBEdi**t с телефонными номерами, чтобы пользователю не пришлось делать это самому.

17. В процедуре нажатия на кнопку **Сохранить и выйти** код простой:

```
procedure TfEditor.Button1Click(Sender: TObject);
begin
  if fDM.TLichData.Modified then fDM.TLichData.Post;
  if fDM.TDoljnost.Modified then fDM.TDoljnost.Post;
  if fDM.TAdres.Modified then   fDM.TAdres.Post;
  if fDM.TTelephones.Modified then  fDM.TTelephones.Post;
  Close;
end;
```

18. У нас осталась кнопка **Добавить сотрудника**. Что мы должны сделать, если пользователь нажмет на эту кнопку? Добавить новую запись в каждую таблицу и перевести курсор в первый **DBEdit**, в котором редактируется фамилия. Это и делаем:

```
procedure TfEditor.Button3Click(Sender: TObject);
begin
  fDM.TLichData.Append;
  fDM.TDoljnost.Append;
  fDM.TAdres.Append;
  fDM.TTelephones.Append;
  DBEdit1.SetFocus;
end;
```

19. Переходим на главную форму. Не забывайте время от времени сохранять проект. Если вы еще не подключили модуль **Editor** к главной форме командой **File - Use Unit**, то сделайте это сейчас, чтобы можно было вызывать окно редактора из главной формы. Начнем с кнопки **Новый сотрудник**. Как и в предыдущем примере, нам потребуется добавить новую запись в каждую таблицу, после чего открыть окно редактора:

```
procedure TfMain.BitBtn3Click(Sender: TObject);
begin
  fDM.TLichData.Append;
  fDM.TDoljnost.Append;
  fDM.TAdres.Append;
  fDM.TTelephones.Append;
  fEditor.ShowModal;
end;
```

20. Сгенерируйте процедуру **onClick** для кнопки **Редактировать**. Тут будет лишь одна строчка кода: **fEditor.ShowModal;**
21. В результате откроется окно редактора, и компоненты будут отображать данные текущей записи. Предположим, пользователю будет удобней дважды щелкнуть по записи в верхней сетке **DBGrid**, чем нажимать кнопку. Поэтому выделите сетку с главной таблицей и сгенерируйте для нее событие **onDBLClick**. Там введите такую же строчку кода.
22. Перейдем к программированию радиокнопок. По нашему замыслу, при открытии программы в верхней сетке **DBGrid** будут отображаться данные из главной таблицы, а в нижней - из таблицы **Adres**. Также будет выделена радиокнопка с надписью **Адрес**. Если пользователю захочется посмотреть должность или телефоны текущего сотрудника, он будет щелкать соответствующую радиокнопку, и эти данные должны быть отображены в нижней **DBGrid**. Выделите первую

радиокнопку с надписью **Адрес** и сгенерируйте для нее событие **onClick**, которое будет возникать, когда пользователь щелкнет по ней. В процедуре этого события впишите следующий код: **if RadioButton1.Checked then DBGrid2.DataSource := fDM.DSAdres;**

Здесь мы проверили, включена ли данная радиокнопка. Если да, то мы меняем связь нижней сетки **DBGrid** и подключаем ее к таблице **Adres**. Ведь связь сетки с таблицей осуществляется через соответствующий компонент **DataSource**, а у нас их четыре. Подключаясь то к одному, то к другому **DataSource**, мы можем программно менять отображенную в сетке таблицу.

23. Для события **onClick** радиокнопки с надписью **Телефоны** код будет таким: **if RadioButton2.Checked then DBGrid2.DataSource := fDM.DSTelephones;**
24. Для события **onClick** радиокнопки с надписью **Должность**, соответственно, код будет следующим: **if RadioButton3.Checked then DBGrid2.DataSource := fDM.DSDoljnost;**
25. Сохраните проект и скомпилируйте его. Запустите проект и заполните таблицы базы данных данными десяти сотрудников.

Внеаудиторная самостоятельная работа:

Составить сравнительную характеристику механизма доступа к данным **Borland Database Engine (BDE)** и механизма доступа к данным **ActiveX Data Object (ADO)**.

Практическая работа №11

Поиск, фильтрация и индексация таблиц. Последовательный перебор, метод Locate, метод Lookup. Фильтрация данных. Использование индексов

Цель работы: изучить особенности механизма поиска и фильтрации данных, а также использования индексов для сортировки; сформировать умения по организации процедур поиска, фильтрации и сортировки данных.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, интегрированная среда разработчика **Turbo Delphi**, система управления базами данных **MS Access 2007**.

Задание 1. Организация поиска данных

Методические указания по выполнению задания:

1. Откройте проект **OK.dpr.**
2. В программах, работающих с базами данных, часто используют поиск данных. Самый простой, но и самый медленный поиск, это последовательный перебор. Вы переходите на первую запись таблицы, создаете цикл, который длится до последней записи, и внутри этого цикла проверяете необходимое условие. Также можно делать и обратный перебор, от последней записи к первой. Приведенный вариант организации поиска нужной записи допустим, если в таблице имеется не более сотни-другой записей, а условная проверка достаточно сложна. Но обычно программисты этот способ не используют, или используют только в крайнем случае. Далее рассмотрим другие способы поиска.
3. Метод **Locate** ищет первую запись, удовлетворяющую условию поиска. Если запись найдена, метод делает ее текущей и возвращает **True**. В противном случае метод возвращает **False** и курсор не меняет положения. Поле, по которому ведется поиск, не обязательно должно быть индексировано. Однако если поле индексировано, то метод ищет запись по индексу, что значительно ускоряет поиск. Поиск может вестись как по одному полю, так и по нескольким полям. Метод имеет три параметра:

   function Locate (const KeyFields: String; const KeyValues: Variant;
   Options: TLocateOptions) : Boolean;

   Параметр **KeyFields** задает поле или список полей, по которым ведется поиск. Если имеется несколько полей, их разделяют точкой с запятой.

Параметр **KeyValues** является вариантным массивом, в котором задаются критерии поиска. При этом первое значение **KeyValues** ставится в соответствие с первым полем, указанным в **KeyFields**. Второе - со вторым, и так далее.

Третий параметр **Options** позволяет задать некоторые опции поиска:

- **loCaseInsensitive** - поиск ведется без учета высоты букв, то есть, считаются одинаковыми строки «строка», «Строка» или «СТРОКА»;
- **loPartialKey** - запись будет удовлетворять условию, если ее часть содержит искомый текст, то есть, если мы ищем «ст», то удовлетворять условию будут «строка», «станция», «стажер» и т.п.

Пустой набор **[]** указывает, что настройки поиска игнорируются. То есть, строка ищется «как есть».

4. При установке компонента **ADOTable** в раздел **uses** прописывается модуль **ADODB**, который содержит описания всех свойств, методов и событий компонента. Желательно использовать метод в том модуле, где установлен этот компонент.
5. Перейдите на модуль **DM**, где у нас хранятся компоненты доступа к базе данных. Процедуру поиска реализуем в этом модуле, а чтобы с ней можно было работать из других форм, опишем ее в разделе **public**:

```
public
  { Public declarations }
  procedure MyLocate(s: String);
```

6. Как видите, в процедуру передается параметр - строка. В ней мы будем передавать искомую фамилию. Если курсор находится на описании нашей процедуры, то нажмите **Ctrl + Shift + C**, чтобы сгенерировать процедуру автоматически. Процедура будет иметь следующий код:

```
procedure TfDM.MyLocate(s: String);
begin
  TLichData.Locate('Фамилия', s, [loPartialKey])
end;
```

Таким образом, при нахождении подходящей записи курсор будет перемещаться к ней.

7. На главной форме выделите компонент **Edit**, предназначенный для поиска по фамилии. Создайте для него событие **onChange**, которое наступает при изменении текста в поле компонента. В созданной процедуре пропишите вызов поиска:

```
procedure TfMain.Edit1Change(Sender: TObject);
begin
fDM.MyLocate(Edit1.Text);
end;
```

Метод **Locate** рекомендуется использовать везде, где это возможно, поскольку он всегда пытается применить наиболее быстрый поиск. Если поле индексировано, и использование индекса ускорит процесс поиска, **Locate** использует индекс. Если поле не имеет индекса, **Locate** все равно ищет данные наиболее быстрым способом. Это делает вашу программу независимой от индексов.

8. Сохраните проект, скомпилируйте и опробуйте результаты поиска.
9. Метод **Lookup**, в отличие от **Locate**, не меняет положение курсора в таблице. Вместо этого он возвращает значения некоторых ее полей. Причем в отличие от **Locate**, этот метод осуществляет поиск лишь на точное соответствие. Такой способ поиска востребован реже, однако в иных случаях этим методом очень удобно пользоваться. Рассмотрим синтаксис этого метода.

   function Lookup (const KeyFields: String;
   const KeyValues: Variant;
   const ResultFields: String) : Variant;

   Как вы видите, первые два параметра такие же, как у **Locate**. А вот третий параметр и возвращаемое значение отличаются. В строке **ResultFields** через точку с запятой перечисляются поля таблицы, значения которых метод должен вернуть. Возвращаются эти значения в виде вариантного массива. Проблема в том, что вернуться может значение **Null**, то есть, ничего, или **Empty** (пустой) и это нужно проверять. Рассмотрим работу метода **Lookup** на примере нашей программы.
10. Прежде всего, вспомним, как работает тип данных **Variant.** В переменную типа **Variant** можно поместить любое значение, в том числе и массив. Этот тип данных обычно используют, когда не известно заранее, данные какого типа нам понадобятся на этапе выполнения программы. Когда переменной типа **Variant** присвоено значение, имеется возможность проверить тип данных этого значения. Для этого служит функция **VarType(): function VarType(const V: Variant): TVarType;**

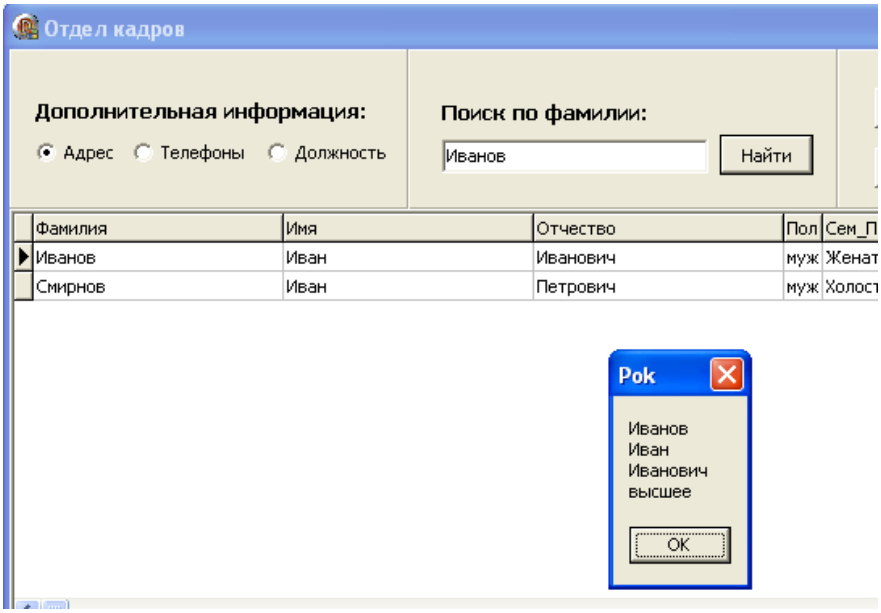
    В качестве параметра в функцию передается переменная вариантного типа. Функция возвращает значение типа **TVarType**. Это значение указывает, какого типа данные содержаться в переменной. Значение может быть **varSmallint** (короткое целое), **varInteger** (целое), **varCurrency** (денежный формат) и так далее. Чтобы увидеть полный

список возвращаемых функцией значений, в редакторе кода установите курсор на название функции и нажмите **Ctrl + F1**, вызвав контекстный справочник. Нас же в данном примере интересуют всего два значения: **varNull** (записи нет) и **varEmpty** (запись пустая). Если в программе мы заранее не проведем проверку на эти значения, то вполне можем вызвать ошибку программы. Если же поиск прошел успешно, то будет возвращен массив вариантных значений, элементы которого начинаются с нуля. Каждый элемент массива будет содержать данные одного из указанных полей.

11. Для поиска воспользуемся кнопкой с надписью «**Найти**», расположенной в верхней части главной формы. Идея такова: пользователь вводит в поле **Edit1** какую то фамилию и нажимает кнопку «**Найти**». Событие **onClick** этой кнопки собирает в строковую переменную значения четырех указанных полей найденной записи. Причем после каждого значения в строку добавляется символ **«#13»** (переход на новую строку), формируя многострочный отчет. Затем эту строку мы выведем на экран функцией **ShowMessage()**.
12. В окне главной формы дважды щелкните по кнопке «**Найти**», генерируя событие **onClick**. Полный код процедуры приведен ниже:

```
procedure TfMain.BitBtn1Click(Sender: TObject);
var
  myLookup: Variant; //для получения результата
  s : String; //для отчета
begin
  //получаем результат:
  myLookup := fDM.TLichData.Lookup('Фамилия', Edit1.Text,
              'Фамилия;Имя;Отчество;Образование');
  //проверяем, не Null ли это:
  if VarType(myLookup) = varNull then
     ShowMessage('Сотрудник с такой фамилией не найден!')
  else if VarType(myLookup) = varEmpty then
     ShowMessage('Запись не найдена!')
  //если это массив, то из его элементов собираем
  //многострочную строку:
  else if VarIsArray(myLookup) then begin
     s := myLookup[0] + #13 +  myLookup[1] + #13 +
            myLookup[2] + #13 + myLookup[3];
     //и выводим ее на экран:
     ShowMessage(s);
  end; //else if
end;
```

13. Сохраните проект, скомпилируйте его и запустите. Опробуйте этот способ поиска.



Задание 2. Фильтрация данных

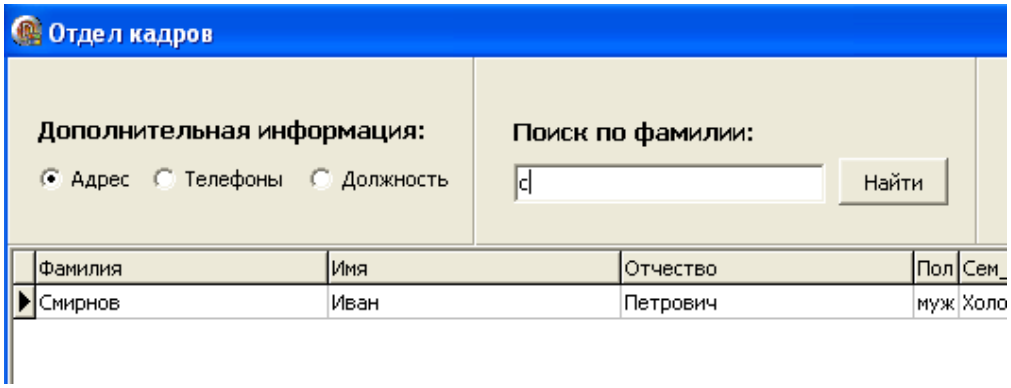
Методические указания по выполнению задания:

1. Фильтрацию данных применяют чаще, чем поиск. Разница в том, что при поиске данных пользователь видит все записи таблицы, при этом курсор либо переходит к искомой записи, либо он получает данные этой записи в виде результата работы функции. При фильтрации дело обстоит иначе. Пользователь в результате видит только те записи, которые удовлетворяют условиям фильтра, остальные записи становятся скрытыми. Конечно, таким образом искать нужные данные проще.
2. Свойство **Filter** - наиболее часто используемый способ фильтрации записей, имеет тип **String**. Вначале программист задает условия фильтрации в этом свойстве, затем присваивает логическому свойству **Filtered** значение **True**, после чего таблица будет отфильтрована. Условия фильтрации должны входить в строку. По правилам синтаксиса, если внутри строки встречается апостроф, его нужно дублировать.
3. Применяя это свойство, достаточно сложных условий задать невозможно, но если условия фильтрации просты, то данный способ незаменим. Опробуем фильтрацию записей на примере нашего приложения. Откройте событие **onChange** компонента **Edit1**, изменим его немного. Закомментируйте или удалите вызов процедуры поиска **MyLocate**, и впишите следующий код:

```
procedure TfMain.Edit1Change(Sender: TObject);
begin
//fDM.MyLocate(Edit1.Text); - закомментировали
  fDM.TLichData.Filter := 'Фамилия >=' + QuotedStr(Edit1.Text);
  fDM.TLichData.Filtered := True;
end;
```

Функция **QuotedStr()** возвращает переданный ей текст, заключенный в апострофы.

4. Откомпилируйте проект и запустите его на выполнение.



5. Событие **onFilterRecord** возникает при установке значения **True** в свойстве **Filtered**. Применение этого способа имеет большой плюс, и большой минус. Плюс в том что, сгенерировав это событие, программист получает возможность задать гораздо более сложные условия фильтрации. Минус же заключается в том, что проверка заключается перебором всех записей таблицы. Если таблица содержит очень много записей, процесс фильтрации может затянуться. В событие передаются два параметра. Первый параметр - набор данных **DataSet**. С ним можно обращаться, как с именем фильтруемой таблицы. Второй параметр - логическая переменная **Accept**. Этой переменной нужно передавать результат условия фильтра. Если условие возвращает **False**, то запись не принимается, и не будет отображаться. Соответственно, если возвращается **True**, то запись принимается.
6. Необходимо отфильтровать записи по начальным (или всем) буквам фамилии, вводимым пользователем в поле **Edit1**. В предыдущем примере, если бы мы ввели букву «И», то вышли бы фамилии, первой буквой которых были бы «И». Это не так удобно. Сделаем так, чтобы если пользователь введет букву «И», то останутся только фамилии, начинающиеся на «И». Если пользователь введет еще букву «в», то останутся только фамилии, начинающиеся на «Ив», и так далее. Поочередно вводя начальные буквы, пользователь доберется до нужных фамилий.

7. Для начала подготовим модуль данных. В нем нам потребуется создать глобальную переменную **ed**, чтобы мы могли передавать в нее текст из компонента **Edit1**:

```
var
  fDM: TfDM;
  ed: String;
```

Этого действия можно было бы избежать, если бы компонент **ADOTable**, подключенный к таблице **LichData**, располагался на главной форме. Но поскольку он находится в модуле данных, то и событие **onFilterRecord** будет сгенерировано в нем. А в этом событии нам нужно будет знать, что в данный момент находится в поле ввода **Edit1**. Именно для этого и нужна глобальная переменная **ed**.

8. Далее выделяем **TLichData**, то есть, компонент **ADOTable**, подключенный к таблице **LichData**. На вкладке **Events** инспектора объектов найдите событие **onFilterRecord** и дважды щелкните по нему, сгенерировав процедуру.

```
procedure TfDM.TLichDataFilterRecord(DataSet: TDataSet; var Accept: Boolean);
var
  s : String; //для значения поля
begin
  //получаем столько начальных букв из поля Фамилия,
  //сколько букв имеется в переменной ed:
  s := Copy(DataSet['Фамилия'], 1, Length(ed));
  //делаем проверку на совпадение значений:
  Accept := s = ed;
end;
```

Здесь в переменную **s** попадает столько начальных букв из поля «**Фамилия**», сколько букв содержит в данный момент компонент **Edit1** на главной форме. Если текст в переменной **s** совпадает с текстом из поля **Edit1**, то переменной **Accept** присваивается **True**, и запись принимается. Иначе запись отфильтровывается. Не забудьте сохранить проект.

9. Далее перейдем в главную форму. Нужно удалить весь текст из события **onChange** компонента **Edit1**, и вписать новый:

```
procedure TfMain.Edit1Change(Sender: TObject);
begin
  //если в поле Edit1 есть хоть одна буква,
  if Edit1.Text <> '' then begin
    fDM.TLichData.Filtered := False; //отключаем фильтр
    ed := Edit1.Text; //передаем в fDM новый текст
    fDM.TLichData.Filtered := True; //включаем фильтр
  end
  //если букв нет, фильтрацию отключаем:
  else fDM.TLichData.Filtered := False;
end;
```

Как только пользователь введет хоть одну букву, срабатывает событие **onChange** компонента **Edit1**. Если в **Edit1** есть хоть одна буква, то мы вначале отключаем фильтрацию, отменяя прошлый фильтр, если он был. Затем мы передаем в глобальную переменную **ed**, расположенную в модуле данных, текст из **Edit1**. Далее снова включаем фильтр. При этом срабатывает событие **onFilterRecord** нашей таблицы, и в этом событии сравнивается текущее значение переменной **ed** и записей поля «**Фамилия**».

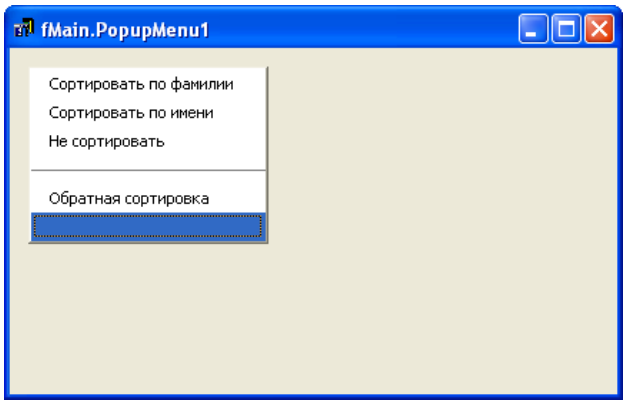
10. Сохраните проект, скомпилируйте и запустите программу. Проверьте, как фильтруются записи.

Задание 3. Сортировка данных

Методические указания по выполнению задания:

1. Создание индексных полей обеспечивает сортировку данных по этим полям, что также облегчает поиск данных - ведь найти нужную фамилию или имя проще, если они отсортированы по алфавиту. Причем имеется возможность сортировать записи не только по возрастанию, но и по убыванию, хотя в большинстве руководств по **Delphi** эта возможность не описывается.
2. При создании в базе данных таблицы **LichData** мы указали поля «Фамилия» и «Имя», как индексированные. Этим и воспользуемся. Чтобы включить сортировку записей по полю «Фамилия», достаточно указать название поля в свойстве **IndexFieldNames** таблицы. Если требуется отключить сортировку, этому свойству присваивается пустая строка.
3. При индексировании таблицы к имени поля можно прибавить строку «**ASC**», если мы желаем сортировать в возрастающем порядке (по умолчанию), или «**DESC**», если сортируем в убывающем порядке. Сортировка «**ASC**» используется по умолчанию.

4. Добавим возможность сортировки по фамилии и имени в нашу программу. Для этого на главную форму установим компонент **TPopupMenu** с вкладки **Standard** палитры компонентов, данный компонент позволяет создавать всплывающие меню, которое появляется по щелчку правой кнопки мыши на объекте, к которому привязано данное меню. Дважды щелкните по компоненту, чтобы открыть редактор меню. Создайте следующие пункты: **Сортировать по фамилии, Сортировать по имени, Не сортировать, -, Обратная сортировка**.



5. В редакторе меню выделите пункт **Сортировать по фамилии** и измените свойство **Name** этого пункта на **NFam**. Пункт **Сортировать по имени** переименуйте в **NImya**. Пункт **Не сортировать** - в **NNet**, а пункт **Обратная сортировка** - в **NObrat**.
6. Вначале создайте обработчик событий для пункта **Не сортировать** (дважды щелкните по пункту) и в заготовку процедуры вставьте следующий программный код:

```
procedure TfMain.NNetClick(Sender: TObject);
begin
  fDM.TLichData.IndexFieldNames := '';
end;
```

7. Для обработчика событий пункта Сортировать по фамилии код немного сложней:

```
procedure TfMain.NFamClick(Sender: TObject);
var
  stype : String;
begin
  //выбираем направление сортировки:
  if NObrat.Checked then stype := ' DESC' //обратная сортировка
  else stype := ' ASC';  //прямая сортировка
  //сортируем
  fDM.TLichData.IndexFieldNames := 'Фамилия' + stype;
end;
```

Здесь, в зависимости от состояния свойства **Checked** пункта **Обратная сортировка** мы присваиваем строковой переменной **stype** либо значение **ASC** (прямая сортировка), либо **DESC** (обратная сортировка). Обратите внимание, что перед строкой имеется

пробел, он нужен, чтобы строка не «прилепилась» к названию поля. Далее мы устанавливаем индекс, указывая имя поля и добавляя к нему значение переменной **stype.** Таким образом, если **Checked** пункта **Обратная сортировка** имеет значение **True** (галочка установлена), мы добавляем **DESC**, или **ASC** в противном случае. В результате имя индексного поля может быть либо «**Фамилия ASC**», либо «**Фамилия DESC**».

8. Сортировку по имени кодируем аналогичным образом:

```
procedure TfMain.NImyaClick(Sender: TObject);
var
  stype : String;
begin
  //выбираем направление сортировки:
  if NObrat.Checked then stype := ' DESC'
  else stype := ' ASC';
  //сортируем
  fDM.TLichData.IndexFieldNames := 'Имя' + stype;
end;
```

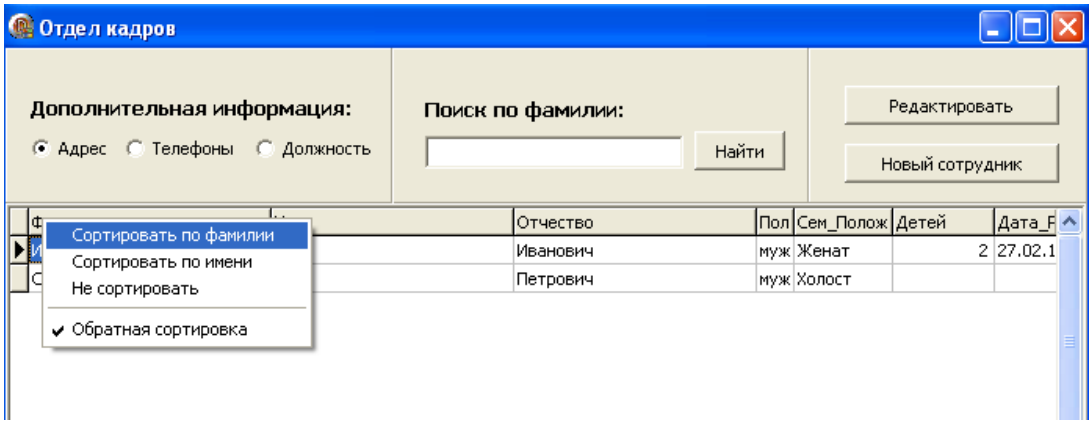
9. Нам осталось указать код пункта всплывающего меню **Обратная сортировка**. Тут нам нужно не просто установить галочку, если ее не было, но также проверить - есть ли сортировка по какому-либо полю. Если таблица отсортирована, требуется ее пересортировать по этому же полю, но уже в обратном порядке.

```
procedure TfMain.NObratClick(Sender: TObject);
begin
//изменяем направление сортировки
  NObrat.Checked := not NObrat.Checked;
  //если сортировка по фамилии, пересортируем
  if Pos('Фамилия',fDM.TLichData.IndexFieldNames)>0 then
    fMain.NFamClick(Sender);
  //если сортировка по имени, пересортируем
  if Pos('Имя',fDM.TLichData.IndexFieldNames)>0 then
    fMain.NImyaClick(Sender);
end;
```

Как видите, мы использовали функцию **Pos()**, которая возвратит ноль, если в строке не найдено указанной подстроки, или номер символа, с которого эта подстрока начинается, если она есть. Нам нужно определить, не входит ли в имя индексного поля «Фамилия» или «Имя».

10. Следует заметить, что при большом количестве записей в таблице смена индексного поля будет несколько замедлять работу приложения. Тем не менее, индексация таблицы - очень удобный и часто применяемый способ организации вывода записей.

11. В свойстве **PopupMenu** верхней сетки **DBGrid1** выберите созданное только что всплывающее меню, чтобы оно открывалось только над этой сеткой, сохраните проект, скомпилируйте его и опробуйте сортировку данных.



12. У нас имеется возможность применить одновременно и фильтрацию записей, и их индексацию. Это позволяет нам создать достаточно мощный и удобный для пользователя механизм поиска записей в нашей программе.

Внеаудиторная самостоятельная работа:

Составить сравнительную характеристику методов поиска, фильтрации и сортировки механизма доступа к данным BDE и механизма доступа к данным ADO.

Практическая работа №12

Основные свойства, события и методы набора данных. Компоненты TADOTable, TADOQuery или TADOStoredProc. Курсоры в наборах данных ADO.

Цель работы: изучить основные свойства, события и методы набора данных. Сформировать умения по организации компонентов TADOTable, TADOQuery или TADOStoredProc. Курсоры в наборах данных ADO..

Задание. Выполнить примеры: Основные свойства, события и методы набора данных. Компоненты TADOTable, TADOQuery или TADOStoredProc. Курсоры в наборах данных ADO.

Свойства

Active - Свойство имеет *логический тип* и позволяет открыть или закрыть *набор данных*, если свойству присвоить True или Falseсоответственно. В зависимости от свойства *CanModify* данные можно либо только просматривать, либо можно также редактировать их.

AutoCalcFields - Свойство логического типа. Если установить *значение* False, то возникновение события *OnCalcFields* будет подавляться, вычисляемые поля обрабатываться не будут. *Значение* True разрешает расчет вычисляемых полей.
Bof - Свойство имеет *логический тип* и содержит True, если *курсор* находится на первой записи набора данных, и False в противном случае. *Bof* содержит True, когда:

- Не пустой *набор данных* открывается.
- При вызове метода *First*.
- При вызове метода *Prior*, если курсор при этом на первой записи набора данных.
- При вызове метода SetRange в пустом наборе данных или диапазоне.

Bookmark - Свойство позволяет установить закладку на текущей записи набора данных. Количество закладок может быть неограниченно, работа с закладками рассматривалась на курсе "Введение в *программирование* на Delphi ". Свойство имеет тип TBookmarkStr.
CanModify - Свойство имеет *логический тип*, и показывает, можно ли редактировать полученный *набор данных*, или он доступен только для чтения. При открытии набора данных автоматически запрашивается *доступ* для редактирования. В таком доступе может быть отказано по разным причинам, например, *таблица* открыта другим пользователем в эксклюзивном режиме. В этом случае *CanModify* получает *значение*False, и мы можем только просматривать данные, но не вносить в них изменения.
DatabaseName - Свойство строкового типа, содержит *адрес базы данных* или ее *псевдоним*. Однако это справедливо к наборам данных *BDE*. В случае использования механизма ADO, это свойство недоступно - вместо него для подключения к базе данных следует использовать свойство Connection или ConnectionString.
DataSource - Свойство используется в наборах данных для указания детального набора данных в отношениях *один-ко-многим*.
DefaultFields - Свойство логического типа, содержит True, если программист не создал ни одного поля в *редакторе полей* набора данных. В этом случае все поля определяются автоматически, в соответствии с данной таблицей.
Eof - Свойство, противоположное свойству *Bof*. Имеет *логический тип*, и имеет *значение* True в случаях, когда:

- Открыт пустой *набор данных*.
- Вызван метод *Last*.
- Вызван метод *Next*, если указатель при этом находится на последней записи таблицы.
- При вызове метода SetRange в пустом наборе данных или диапазоне.

FieldCount - Свойство целого типа, содержит количество полей в наборе данных.
Fields - Свойство позволяет получить *значение* нужного поля по его индексу. Поля при этом индексируются с нуля. Например, получить *значение* седьмого по счету поля набора данных можно так:

```
Edit1.Text := CustTable.Fields[6].Value;
```

FieldValues - Свойство позволяет получить *значение* нужного поля по его имени. Это свойство используется по умолчанию, поэтому его можно не указывать. Примеры:

```
Edit1.Text := CustTable. FieldValues ['Order'];
Edit1.Text := CustTable['Order'];
```

Filter - Свойство строкового типа. Содержит строку, которая определяет правила фильтрации набора данных.
Filtered - Свойство логического типа. Если в свойстве *Filter* имеется строка, определяющая порядок фильтрации, то присвоение значения True свойству *Filtered* приводит к фильтрации набора данных. Присвоение этому свойству False отменяет фильтрацию.

FilterOptions - Свойство имеет тип TFilterOptions и применяется для строковых или символьных полей. Свойству можно
присвоить *значение* foCaseInsensitive или foNoPartialCompare. В первом случае фильтрация будет учитывать *регистр* букв, во втором учитывается лишь *точное совпадение* образцу.
Modified - Очень важное свойство логического типа. Содержит True, если *набор данных* был изменен, и False в противном случае. Часто применяется для проверок: если *набор данных* изменен, то вызвать метод *Post*, чтобы сохранить изменения.
RecNo и *RecordCount* - Свойства целого типа. Первое содержит номер текущей записи в наборе данных, второе - общее количество записей.
State - Очень важное свойство, определяющее состояние набора данных. Может иметь следующие значения:

- dsInactivate - *набор данных* закрыт.
- dsBrowse - режим просмотра.
- dsEdit - режим редактирования.
- dsInsert - *режим вставки*.
- dsSetKey - поиск записи.
- dsCalcFields - состояние установки вычисляемых полей.
- dsFilter - режим фильтрации записей.
- dsNewValue - режим обновления свойства TField.NewValue.
- dsOldValue - режим обновления свойства TField.OldValue.
- dsCurValue - режим обновления свойства TField.CurValue.
- dsBlockRead - состояние чтения блока записей.
- dsInternalCalc - обновление полей, у которых свойство FieldKind соответствует значению fkInternalCalc.

Методы
Append - Метод добавляет новую *запись* в конец набора данных. При этом *набор данных* автоматически переходит в режим редактирования.
***AppendRecord* (const Values: array of const)** - Метод добавляет новую *запись* в конец набора данных, и заполняет поля этой записи значениями из массива, переданного в метод как *параметр*.
Cancel - Отменяет все изменения набора данных, если они еще не сохранены методом *Post* или переходом на другую *запись*.
ClearFields - Метод очищает все поля текущей записи.
Close - Закрывает *набор данных*. Метод является альтернативой присваивания False свойству *Active* набора данных.
Delete - Метод удаляет текущую *запись*. Следует заметить, что во многих форматах данных удаляемая *запись* лишь помечается, как удаленная, и скрывается от пользователя. Физически же такая *запись* из файла не удаляется. В этом случае обычно время от времени приходится "паковать" таблицы, избавляясь от таких записей.
Edit - Метод переводит *набор данных* в состояние редактирования. Если этого не сделать, *изменение записи* будет невозможным.
FieldByName - Еще один способ получить *значение* поля или изменить его, указывая имя поля. При этом можно использовать *явное преобразование* данных в нужный тип, например, AsInteger, AsString и т.п. Пример:
Table1. FieldByName ('QUANTITY').AsInteger := StrToInt(Edit1.Text);
FindFirst , *FindLast* , *FindNext* и *FindPrior* - Методы пытаются установить *курсор* соответственно, на первую, на последнюю, на следующую и на

предыдущую *запись*. В случае успеха методы возвращают True. Переход к другой записи приводит к автоматическому сохранению изменений, если изменения были.
First **,** *Last* **,** *Next* и *Prior* - просто устанавливают *указатель* соответственно на первую, последнюю, следующую и предыдущую *запись*. Переход к другой записи приводит к автоматическому сохранению изменений, если изменения были.
FreeBookmark - Метод освобождает *память*, связанную с закладкой *Bookmark*. Обычно вместо вызова этого метода достаточно присвоить закладке пустую строку (см. лекцию 30 курса "Введение в *программирование* на Delphi").
GotoBookmark - Метод обеспечивает переход на закладку *Bookmark*, переданную в качестве параметра.
Insert - Метод вставляет новую *запись* в указанную в параметре позицию набора данных. При этом *набор данных* автоматически переходит в режим редактирования.
InsertRecord **(const Values: array of const)** - Метод вставляет новую *запись* в *набор данных*, и заполняет поля этой записи значениями из массива, переданного в метод как *параметр*. Пример:

```
Customer. InsertRecord ([CustNoEdit.Text, CoNameEdit.Text,
                         AddrEdit.Text, Null, Null, Null, Null,
                         Null, Null, DiscountEdit.Text]);
```

Обратите внимание, что в некоторые поля были вставлены значения Null, то есть, ничего. То же самое происходит, когда *пользователь* при редактировании записи вносит значения не во все поля.
IsEmpty - Метод возвращает True, если в наборе данных нет записей. Применяется для проверки - не пуста ли *таблица*?
Locate - Метод ищет *запись* в наборе данных (см. предыдущую лекцию).
Lookup - Метод ищет *запись* в наборе данных (см. предыдущую лекцию). В отличие от *Locate* не переводит *указатель* на найденную *запись*, а лишь возвращает значения ее полей.
Open - Метод открывает *набор данных*. То же самое происходит, если свойству *Active* набора данных присвоить *значение* True**.**
Post - Метод сохраняет сделанные изменения в наборе данных.
Refresh - Метод заново перечитывает таблицу и обновляет *набор данных*. Имеет смысл использовать в приложениях, где несколько пользователей работают с одной базой данных.
События
After … - События, возникающие после вызова соответствующего метода:
AfterCancel - Событие возникает после *отмены изменений* в текущей записи.
AfterClose - Событие возникает после закрытия набора данных.
AfterDelete - Событие возникает после удаления текущей записи.
AfterEdit - Событие возникает после перехода набора данных в режим редактирования.
AfterInsert - Событие возникает после вставки новой записи.
AfterOpen - Событие возникает после открытия набора данных.
AfterPost - Событие возникает после вызова метода *Post***.**
AfterScroll - Событие возникает после перехода на другую *запись***.**
Before … - События, возникающие перед вызовом соответствующего метода:
BeforeCancel - Событие возникает перед отменой изменений в текущей записи.
BeforeClose - Событие возникает перед закрытием набора данных.
BeforeDelete - Событие возникает перед удалением текущей записи.
BeforeEdit - Событие возникает перед переходом набора данных в режим редактирования.

BeforeInsert - Событие возникает перед вставкой новой записи.
BeforeOpen - Событие возникает перед открытием набора данных.
BeforePost - Событие возникает перед вызовом метода *Post*.
BeforeScroll - Событие возникает перед переходом на другую *запись*.
OnCalcFields - Событие возникает при необходимости переопределения вычисляемых полей. Такое событие возникает всякий раз, когда *программа* должна сформировать значения для вычисляемых полей. Событие возникает также при открытии набора данных, и при любом его изменении. Если *алгоритм* вычислений достаточно сложен, *база данных* большая, а *пользователь* интенсивно с ней работает, событие *OnCalcFields* может значительно замедлить работу с базой данных. В этом случае следует отключать это событие. Для этого достаточно
присвоить *значение* False свойству *AutoCalcFields* текущего набора данных.
OnFilterRecord - Событие возникает при включении фильтрации записей.^  p>
OnNewRecord - Событие возникает при вызове методов *Append* или *Insert*.
Блокировка таблиц в архитектуре файл-сервер
При работе в *архитектуре файл-сервер* с единой сетевой базой данных работают несколько клиентских приложений. При этом нередко возникает ситуация, когда один *пользователь* вносит изменения в базу данных. В этот момент, во избежание потери или порчи данных, следует запретить внесение изменений другими пользователями. Такая *блокировка* достигается методом ***LockTable*** **()**. При
этом *значение* свойства *LockType* этого набора данных определяет вид запрета. *Значение* ltReadLock запрещает чтение, а ltWriteLock - *запись* в *набор данных*. Можно запретить и чтение, и *запись*, но для этого следует вызвать
метод *LockTable* () дважды. *Блокировка* таблиц методом *LockTable* () справедлива для таблиц Paradox или dBase, если вы используете механизм *BDE*. Когда изменения внесены, и необходимость блокировки пропадает, можно *снять блокировку* методом *UnlockTable***()**, указав в параметре тип снимаемого запрета (ltWriteLock, ltReadLock).
Свойство ***LockType*** набора данных ADO имеет тип **TADOLockType**:
type TADOLockType = (ltUnspecified, ltReadOnly, ltPessimistic, ltOptimistic, ltBatchOptimistic);
Это свойство позволяет определить тип блокировки при открытии набора данных. Как видно из описания типа, свойство может иметь следующие значения:
ltUnspecified - тип блокировки не определен.
ltReadOnly - *блокировка* записи, читать данные можно.
ltPessimistic - *пессимистическая блокировка*. Свойство указывает, что если вы редактируете *запись*, то другие пользователи не смогут редактировать ее, пока вы не сохраните изменения.
ltOptimistic - *оптимистическая блокировка*. *Блокировка* подразумевает, что возникновение конфликта маловероятно. В связи с этим любой *пользователь* в любое время может редактировать любую *запись*. Проверка на наличие конфликтов производится только в момент сохранения изменений.
ltBatchOptimistic - свойство устанавливает блокировку на пакет записей, а не на отдельную *запись*. При этом все обновления, сделанные пользователем, не записываются сразу, а накапливаются в оперативной памяти. Позже они сохраняются одним пакетом. Такой подход увеличивает *производительность* приложения, но также увеличивается риск возникновения конфликтов.
Курсоры в наборах данных ADO

Наборы данных ADO имеют два специфичных свойства, неразрывно связанные друг с другом: *CursorLocation* и *CursorType*. Курсоры оказывают большое влияние на то, каким образом извлекаются данные из таблиц, каким образом вы можете перемещаться по ним и т.д. **Фактически, курсор - это механизм перемещения по записям набора данных**. От того, какой *курсор* используется в многопользовательской среде, зависит способ перемещения по записям: только вперед или в обе стороны. Будете ли вы видеть изменения, сделанные другими пользователями, также зависит от типа применяемого курсора.

CursorLocation (положение курсора)

Это свойство определяет, каким образом извлекаются и модифицируются данные. Значений только два:

clUseClient - *курсор* на стороне клиента.

clUseServer - *курсор* на стороне сервера.

Клиентский *курсор* обслуживается механизмом ADO Cursor Engine. В момент открытия набора данных все данные перекачиваются с сервера на клиентский *компьютер*. После этого данные хранятся в оперативной памяти. Перемещения по данным и их модификация происходит значительно быстрее, кроме того, клиентский *курсор* обладает более широкими возможностями.

Серверный *курсор* обслуживается операционной системой. Благодаря тому, что *курсор* находится на стороне сервера, приложению нет смысла перекачивать все данные разом, это повышает скорость работы с *БД*. Серверные курсоры больше подходят для обслуживания больших наборов данных. Следует заметить, что если вы работаете с локальной базой данных (например, Access), то серверный *курсор* будет обслуживаться программой, обслуживающей эту базу данных.

CursorType (тип курсора)

Имеется пять типов курсора:

- **Unspecified** - не указанный. В Delphi такой тип не используется, он присутствует только потому, что имеется в ADO.
- **Forward-only** (только вперед). Этот тип курсоров обеспечивает самую высокую производительность, однако он позволяет перемещаться по записям только в одном направлении - от начала к концу, что делает его малопригодным для создания пользовательского интерфейса. Однако он хорошо подходит для программных операций, таких как перебор записей, формирование отчета и т.п.
- **Static** (статический) - пользователь имеет возможность перемещаться в обоих направлениях, однако изменения записей, выполненные другими пользователями, не видны таким курсором.
- **Keyset** (набор ключей). При открытии набора данных с сервера читается полный список всех ключей. Этот набор ключей хранится на стороне клиента. Если приложение нуждается в данных, провайдер OLE DB читает строки таблицы. Однако после открытия набора данных в этот набор нельзя добавлять новые ключи, или удалять имеющиеся. То есть, если другой пользователь добавил новую запись, текущий клиент ее не увидит. Однако он увидит изменения существующих записей, сделанные другими пользователями.
- **Dynamic** (динамический). Самый мощный тип курсора, но при этом и самый ресурсоемкий. Он позволяет видеть все изменения, все добавления или удаления, сделанные другими пользователями, но при этом больше других замедляет работу с БД.

Практическая работа №13

СервербазданныхBorlandInterBase. SQL-серверLocalInterBase. Физическая организация базы данных формата InterBase. Организация сеанса связи с удаленной базой данных. Основы администрирования SQL-сервера BorlandInterBase.

Цель работы: сформировать умения по организации сеанса связи с удаленной базой данных; научится администрировать сервер баз данных **Borland InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

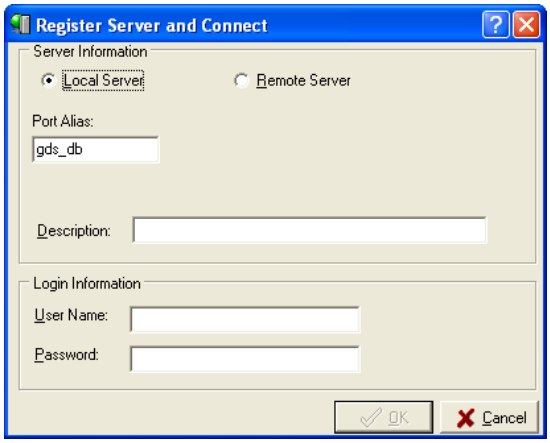
Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**.

Задание 1. Регистрация сервера

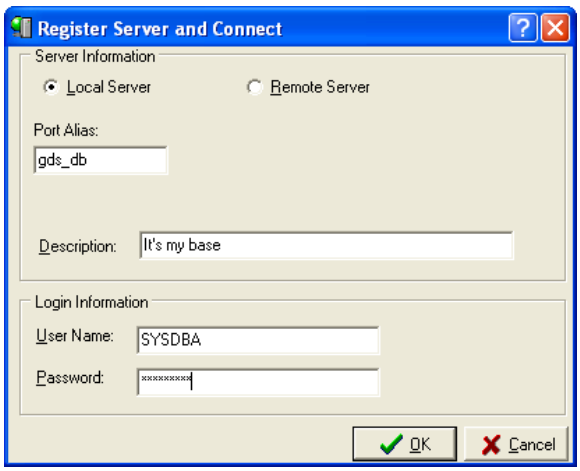
Методические указания по выполнению задания:

- После установки SQL-сервера способ его запуска определяется с помощью специальной программы InterBase Manager, доступ к которой можно получить через команду **Пуск – Все программы – Borland InterBase 7.5 - InterBase Server Manager**.
- Сервер может запускаться автоматически при загрузке операционной системы (переключатель **Automatic**) или же вручную (переключатель **Manual**). В последнем случае для запуска сервера следует щелкнуть на кнопке **Start** в окне **InterBase Manager**.

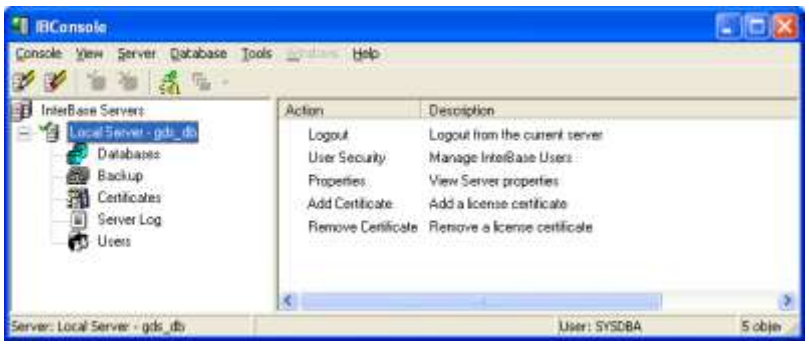
- Все средства администрирования и работы с базами данных **InterBase** (включая редактор SQL-запросов) находятся в одном приложении — **IBConsole**, для запуска которого используется команда **Пуск – Все программы – Borland InterBase 7.5 - IBConsole**.
- В левой части окна **IBConsole** расположен список зарегистрированных серверов **InterBase**. Для регистрации следует выбрать команду **Server - Register** или просто дважды щелкнуть на корневом элементе **InterBase Servers**. В результате на экране появится диалоговое окно **Register Server and Connect**.



- В данном окне следует указать, к какому серверу будет выполнено подключение: локальному (переключатель **Local Server**) или удаленному (переключатель **Remote Server**). Локальным является сервер, доступный через локальный или сетевой диск, а удаленным — сервер, доступ к которому выполняется по имени сетевого ресурса. В этом случае в поле **Server Name** в водится сетевое имя (или сетевой адрес), соответствующее протоколу, изданному в поле **Network Protocol**.
- В разделе **Login Information** указывается имя пользователя и пароль, без которых нельзя подключиться ни к одной базе данных. В среде **InterBase** в качестве имени стандартного пользователя, обладающего полными правами администратора, указывают **sysdba** (поле **User Name**), а в качестве пароля - **masterkey** (поле **Password**). При желании, эти значения в дальнейшем можно изменить.



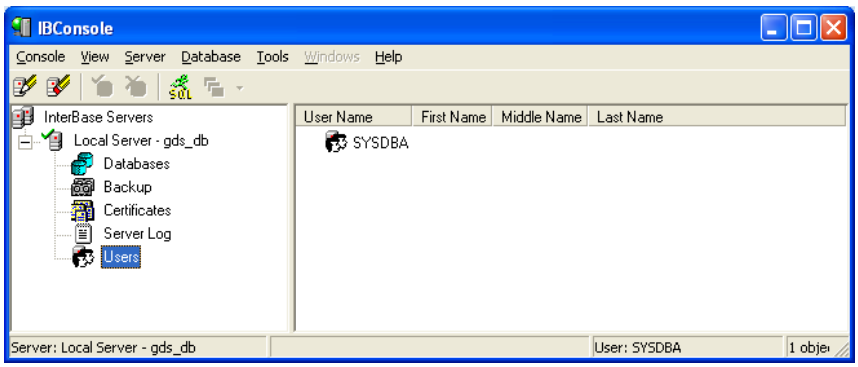
- Если имя пользователя и пароль были введены правильно, то после щелчка на кнопке **ОК** произойдет подключение к выбранному SQL-серверу **InterBase** и у корневой папки **InterBase Servers** появится подпапка. В случае локального подключения к базе данных это будет подпапка **Local Server**. После двойного щелчка на ней отобразится список элементов, используемых при работе с локальным SQL-сервером.



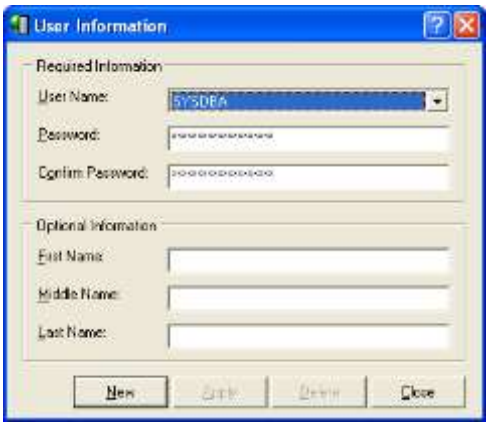
Задание 2. Работа с пользователями

Методические указания по выполнению задания:

- Если в **IBConsole** выбрать элемент **Users**, то в правой части окна отобразится список всех пользователей, зарегистрированных для данного сервера. При отсутствии пользователей всегда будет зарегистрирован только пользователь **SYSDBA**.



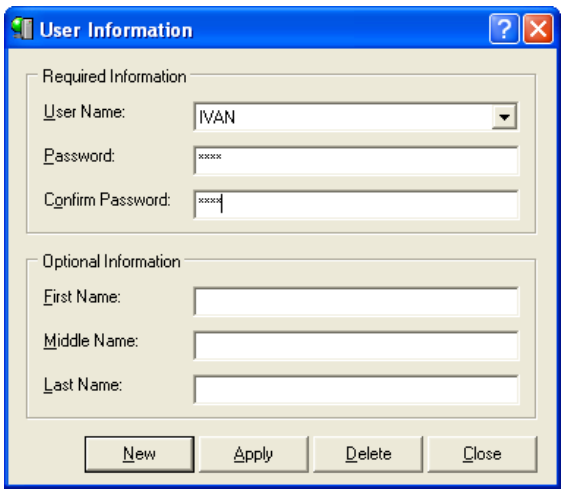
- Этому пользователю соответствует специальная учетная запись, которая позволяет игнорировать систему защиты SQL-сервера и выполнять любые задачи администрирования. После установки **InterBase** пользователь **SYSDBA** — единственный, и он должен зарегистрировать всех остальных пользователей сервера.
- Для администрирования пользователей в программе **IBConsole** используют диалоговое окно **User Information**, которое открывается командой **Server - User Security** или двойным щелчком на имени пользователя. Можно также щелкнуть правой кнопкой мыши на элементе **Users** и для добавления пользователя выбрать в контекстном меню команду **Add User**, а при необходимости изменить учетную запись пользователя — команду **Modify User**.



- Для того чтобы зарегистрировать на сервере нового пользователя, в диалоговом окне **User Information** необходимо выполнить следующие действия:
  - Щелкнуть на кнопке **New**.
  - Ввести имя пользователя в поле **User Name (IVAN)**.
  - Ввести пароль пользователя в полях **Password (USER)** и **Confirm Password (USER)**.
  - При желании ввести дополнительную информацию - фамилию, имя и отчество

пользователя в разделе **Optional Information**.

- Щелкнуть на кнопке **Apply**, чтобы создать новую учетную запись.



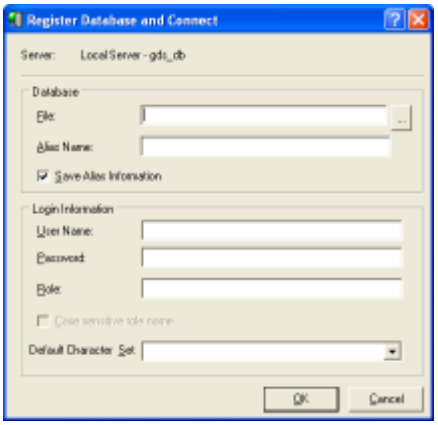
- Для того чтобы изменить информацию о пользователе, в диалоговом окне **User Information** необходимо выполнить следующие действия.
  - Выбрать пользователя в раскрывающемся списке **User Name**.
  - Откорректировать в соответствии с новым выбором содержимое остальных полей.
  - Для сохранения изменений щелкнуть на кнопке **Apply**.
- Имя пользователя изменить нельзя. Единственный способ сделать это — удалить пользователя, а затем создать новую учетную запись с другим именем. Для того чтобы удалить информацию о пользователе, в диалоговом окне **User Information** необходимо выполнить следующие действия:
  - Выбрать пользователя в раскрывающемся списке **User Name**.
  - Щелкнуть на кнопке **Delete**. В результате на экране появится запрос на подтверждение операции. Если в ответ на этот запрос щелкнуть на кнопке **ОК**, то текущий пользователь будет удален и больше не сможет подключиться к базам данных сервера.
- Новые пользователи автоматически не получают прав для изменения или даже просмотра информации, хранимой в базе данных, — права доступа должны быть предоставлены явно. Пользователь (естественно, кроме **SYSDBA**) не сможет получить доступ ни к одному из объектов базы данных до тех пор, пока ему не будут на это предоставлены соответствующие права.
- Права доступа в сервере **InterBase** назначаются при помощи так называемых ролей и

SQL-команды **GRANT**.

Задание 3. Регистрация и просмотр существующей базы данных

Методические указания по выполнению задания:

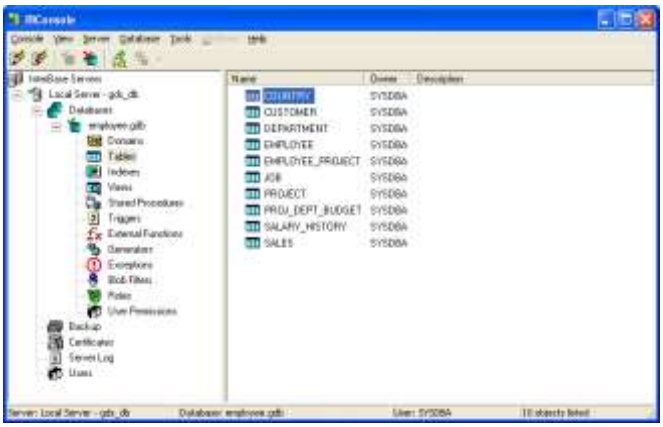
- Существующую базу данных **InterBase** можно зарегистрировать в программе **IBConsole**, щелкнув правой кнопкой мыши на элементе **Databases** и затем выбрав в контекстном меню команду **Register** или выбрав эту же команду в главном меню **Database**.
- В качестве примера зарегистрируем базу данных **employee.gdb**, которая устанавливается вместе с файлами примеров **InterBase** в папку **\Program Files\ Borland\ lnterBase\ examples\Database**.
- После выполнения команды **Register** одним из вышеупомянутых способов откроется диалоговое окно **Register Database and Connect**.



- В поле **Database** следует ввести путь к размещению файла **employee.gdb** (или же выбрать этот файл с помощью диалогового окна поиска файлов), а в поле **File – employee.gdb**.
- В результате у элемента **Databases** в окне **IBConsole** появится подчиненный элемент **employee.gdb**, который в свою очередь имеет собственный набор подчиненных элементов:
    - домены (**Domains**),
    - таблицы (**Tables**),
    - индексы (**Indexes**),
    - представления (**Views**),
    - хранимые процедуры (**Stored Procedures**),

- внешние функции (**External Functions**),
- генераторы (**Generators**),
- исключения (**Exceptions**),
- BLOB-фильтры (**BLOB Filters**),
- роли (**Roles**).

- Для того чтобы просмотреть, например, список таблиц в выбранной базе данных, следует выделить элемент **Tables**.



- Для работы с какой-нибудь таблицей следует дважды щелкнуть на ее имени в списке. В результате откроется диалоговое окно **Properties for**, где на вкладке **Properties** можно просмотреть структуру выбранной таблицы, на вкладке **Metadata** — SQL-команду, которая была использована для создания этой таблицы, на вкладке **Permissions**— права зарегистрированных пользователей на выполнение различных операций, на вкладке **Data** — собственно данные таблицы, а на вкладке **Dependencies** — взаимосвязи текущей таблицы с другими элементами базы данных.

Задание 4. Отключение от базы данных, отмена регистрации и отключение от сервера

Методические указания по выполнению задания:

- Чтобы отключиться от активной базы данных в программе **IBConsole**, ее необходимо выделить в списке **Databases** и в меню **Database** выбрать команду **Disconnect** (или соответствующую команду контекстного меню) Для повторного подключения к зарегистрированной базе данных под именем активного пользователя SQL-сервера служит команда **Connect**, а для подключения с вводом произвольного имени пользователя — команда **Connect As**.
- Для отмены регистрации базы данных (только отключенной) в меню **Database** используется команда **Unregister** (или соответствующая команда контекстного меню). После того как отмена регистрации подтверждена, база данных исчезнет из списка **Databases**. Выполните отмену регистрации базы данных **employee.gdb**
- Чтобы отключиться от активного SQL-сервера, его необходимо выделить в списке **InterBase Servers** и в меню **Server** выбрать команду **Logout** (или соответствующую команду контекстного меню).
- Для отмены регистрации SQL-сервера (только неактивного) в меню **Server** используется команда **Un-Register** (или соответствующая команда контекстного меню). После того как отмена регистрации подтверждена, SQL-сервер исчезнет из списка **InterBase Servers**.

Внеаудиторная самостоятельная работа:

Составить опорный конспект по основным командам работы с сервером баз данных **Borland InterBase**.

Практическая работа №14

Создание и перенос базы данных. Создание базы данных. Регистрация базы данных. Перенос базы данных из локальных БД в InterBase. Типы данных. Домены.

Цель работы: сформировать умения по созданию, регистрации базы данных в **InterBase**; познакомиться с основными типами данных; сформировать умения по созданию доменов.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и

способы выполнения профессиональных задач, оценивать их эффективность и качество.

- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**.

Задание 1. Создание базы данных. Оператор CREATE DATABASE

Методические указания по выполнению задания:

- В общем случае оператор **create database** имеет следующий синтаксис:

  CREATE DATABASE имя_базы_данных

  Здесь имя_базы_данных — полный путь к файлу базы данных, заключенный в одинарные кавычки.
- В различных диалектах SQL синтаксис оператора **CREATE DATABASE** расширен дополнительными обязательными параметрами. Например, в InterBase, кроме имени самого файла базы данных, необходимо обязательно указать имя пользователя и пароль:

  CREATE DATABASE имя базы данных

  USER имя пользователя PASSWORD пароль
- В качестве примера создадим базу данных **staff.gdb** в корневом каталоге диска **D:** от имени системного администратора **sysdba** с паролем **masterkey**:

  CREATE DATABASE 'd:\Staff.gdb'

  USER 'sysdba' PASSWORD 'masterkey'

- Язык SQL нечувствителен к регистру символов, но, тем не менее, по негласным соглашениям, применяемым при оформлении кода SQL-операторов, ключевые слова языка принято записывать прописными буквами.
- В результате выполнения представленного выше оператора на диске **D:** появится файл **staff.gdb** — пустая (т.е. не содержащая таблиц пользователя) база данных **InterBase** с параметрами, выбранными по умолчанию. Под параметрами, в частности, подразумевается предельный размер файла базы данных. В отличие от локальных баз данных, наподобие **dBASE** или **Paradox**, состоящих из набора разнообразных файлов, все объекты клиент-серверных баз данных находятся в одном файле. Размер этого файла ограничен и задается при создании базы данных. Таким образом, если предполагается, что база данных будет большой, предельный размер файла имеет смысл увеличить. Например, в **InterBase** размер базы данных определяется размером и количеством внутренних страниц (блоков данных). Так, для создания базы данных **staff.gdb** с предельным размером в 40 Мбайт следует выполнить SQL-оператор:

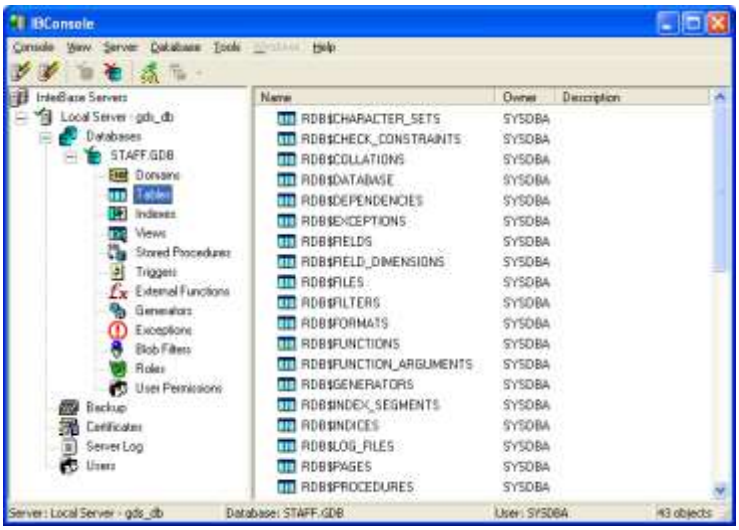
  CREATE DATABASE 'd:\Staff.gdb'
  USER 'sysdba' PASSWORD 'masterkey'
  PAGE_SIZE 4096 LENGTH = 10000 PAGES
- Для того чтобы создать новую базу данных **staff.gdb**, ее старый вариант необходимо удалить с диска. Это можно сделать вручную в любой программе управления файлами, предварительно закрыв окно **Interactive SQL**, чтобы разорвать связь с существующей базой данных.
- По умолчанию в **InterBase** размер страницы составляет 1024 байт, что во многих случаях слишком мало. Оптимальный вариант - 4096 байт, а в случае использования больших таблиц - даже 8192 байт, поскольку увеличение размера страницы повышает быстродействие работы с базой данных.

Задание 2. Системные и пользовательские данные

Методические указания по выполнению задания:

- В **InterBase** (как и в любом другом SQL-сервере) под «пустой» следует понимать базу данных, не содержащую таблиц пользователя. Это очевидно, если зарегистрировать файл **staff.gdb** в программе IBConsole и просмотреть перечень ее таблиц.

- Однако в действительности только что созданный файл базы данных включает множество системных таблиц, содержимое которых используется для организации работы с данными. Именно по этой причине файл «пустой» базы данных имеет размер, далекий от нулевого (в примере с staff.gdb - 680 Кбайт). Для того чтобы просмотреть перечень системных элементов базы данных **InterBase**, в программе **IBConsole** следует выбрать команду **View - System Data**.
- Содержимое системных таблиц изменять настоятельно не рекомендуется.



Задание 3. Основные операторы по работе с базой данных

Методические указания по выполнению задания:

- Для подключения к базе данных используется SQL-оператор CONNECT со следующим синтаксисом:

  CONNECT имя базы данных

  USER имя _пользователя PASSWORD пароль

- Пример для базы данных **staff.gdb**:

CONNECT 'd:\staff.gdb'

USER 'sysdba' PASSWORD 'masterkey'

- Для отключения от базы данных (т.е. скрытия ее файла и освобождения всех занятых ею ресурсов) предназначен SQL-оператор **DISCONNECT**. В различных диалектах SQL этот оператор может отличаться. Например, в **InterBase** для отключения от всех активных баз данных следует выполнить следующую команду: **DISCONNECT ALL**
- Как и другие SQL-серверы, **InterBase** поддерживает создание дополнительных файлов базы данных на тот случаи, если предельный объем основного хранилища исчерпан. Для этого используется SQL-оператор **ALTER DATABASE**. Пример простейшего варианта создания вторичного файла базы данных в синтаксисе **InterBase**:

  ALTER DATABASE ADD FILE 'd:\staff.gd1'

  Оператор **ALTER DATABASE** действует для текущей базы данных (предположим, что в данном случае - это staff.gdb).
- В данном примере файл **staff.gd1** был создан с параметрами базы данных, выбранными по умолчанию. Подобно основному файлу, для вторичных файлов в InterBase можно явно задать их предельный размер в страницах с помощью ключевого слова **LENGTH** (размер страницы определяется по установке, выполненной для основного файла):

  ALTER DATABASE

  ADD FILE 'd:\staff.gd1'

  LENGTH = 10000 PAGES
- Конечно, в простейших случаях файл базы данных можно удалить и вручную, с помощью любой программы управления файлами, но нечаянно можно забыть о том, что были созданы вторичные файлы базы данных. Поэтому самый лучший способ удалить активную базу данных со всеми связанными с ней дополнениями - воспользоваться SQL-оператором **DROP DATABASE** (имя базы данных явно не указывается).

Задание 4. Типы данных SQL

Методические указания по выполнению задания:

- Таблица — это совокупность столбцов, каждый из которых хранит данные определенного типа. Типы данных назначаются полям таблицы в момент ее создания. Типы Различными стандартами и диалектами языка SQL поддерживаются следующие

типы данных.

- Типы данных, определенные стандартом SQL1:
  - **CHARACTER(n)** или **CHAR(n)** — строковый тип фиксированной длины в n символов.
  - **DECIMAL(p,s)** — числа с десятичной точкой. Параметру p соответствует общее количество разрядов, а параметру s — количество цифр после запятой. Так, поле типа **DECIMAL(5,2)** может содержать только числа в формате PPP,ee.
  - **DOUBLE PRECISION** — вещественные числа с двойной точностью (относительно чисел типа **FLOAT**).
  - **FLOAT** — числа с плавающей запятой.
  - **INTEGER** или **INT** — длинные целые числа со знаком.
  - **NUMERIC (p, s)** — числа, аналогичные объявленным с помощью типа **DECIMAL**, однако в данном случае они не обязательно должны состоять из количества разрядов, заданного параметром p. Другими словами, поле типа **NUMBER(5,2)** может содержать как 123,78, так и 1,43.
  - **SMALLINT** — короткие целые числа со знаком (диапазон значений - от -32768 до 32767).
- В стандарте SQL2 было добавлено несколько новых типов:
  - **BIT(n)** — строка битов фиксированной длины n;
  - **BIT VARYING(n)** — строка битов переменной длины, но не более чем n;
  - **DATE** — дата;
  - **INTERVAL** — временной интервал;
  - **TIMESTAMP** — дата и время;
  - **VARCHAR(n)** — строковый тип переменной длины, но не более чем n символов.
- Однако в конкретных форматах баз данных некоторые из перечисленных выше типов могут не поддерживаться. Например, при создании локальных таблиц Paradox не используются типы **DECIMAL** и **DOUBLE PRECISION** из стандарта SQ.L1, а также тип **VARCHAR** из стандарта SQL2. В то же время в диалекте SQL **InterBase** отсутствуют типы **BIT, BIT VARYING** и **INTERVAL** из стандарта SQL2.
- Кроме того, во многих форматах используются дополнительные типы, не входящие в стандарты SQL1 и SQL2, однако поддерживаемые конкретным диалектом SQL. Так,

например, как в локальных, так и в клиент-серверных базах данных часто используется тип BLOB. Аббревиатура BLOB означает Binary Large Object, т.е. большой двоичный объект. Поля такого типа могут использоваться для хранения больших текстовых фрагментов, графики, звука и т.д.

- Еще несколько примеров дополнительных типов, поддерживаемых различными диалектами SQL:
  - **AUTOINC** — целочисленные поля с автоматическим приращением значений при добавлении в таблицу новой строки (например, в формате Paradox 7);
  - **BOOLEAN** — булев (логический) тип;
  - **TIME** — время.
- Точный набор поддерживаемых типов можно определить только по техническому описанию конкретной системы управления базами данных.

Типы данных в Borland InterBase

Тип	Размер	Описание
BLOB	Переменный	Двоичные данные любого типа
BOOLEAN	2 байт	Логический тип. Возможные значения: TRUE, FALSE, UNKNOWN
CHAR, CHARACTER	1 байт на символ	Строковый тип фиксированной длины
DATE	4 байт	Дата в диапазоне от 1 января 100 года по 29 февраля 32768 года
DECIMAL	Переменный (2, 4 или 8 байт)	Количество разрядов может составлять от 1 до 18
DOUBLE, PRECISION	8 байт	Вещественные числа в диапазоне от 2,225*10-308 до 1,797*10 308
FLOAT	4 байт	Вещественные числа в диапазоне от 1,175*10-38 до3,402*1038
ITEGER	4 байт	Длинные целые числа в диапазоне от -2147483648 до 2147483647
NUMERIC	Переменный (2, 4 или 8 байт)	Количество разрядов может составлять от 1 до 18
SMALLINT	2 байт	Короткое целое число в диапазоне от-32768 до 32767
TIME	4байг	Время
TIMESTAMP	8 байт	Дата и время
VARCHAR	1 байт на символ	Строковые данные переменной длины

Внеаудиторная самостоятельная работа:

Составить опорный конспект письменно в тетради по основным командам создания, регистрации и переноса базы данных в **InterBase**.

Практическая работа №15

Работа с таблицами. Создание, модификация и удаление таблиц. Изменение данных в таблицах

Цель работы: сформировать умения по выполнению команд создания, модификации и удаления таблиц в **InterBase**; сформировать умения по выполнению команд изменения данных в таблицах **InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**.

Задание 1. Создание таблиц

Методические указания по выполнению задания:

- Рассмотрим пример создания базы данных, содержащей основные сведения о сотрудниках предприятия. Распределим данные между шестью таблицами:
    - **STAFF** - список сотрудников с некоторыми индивидуальными данными;

- **REGIOHS** - список районов в местности размещения предприятия, сформированный в соответствии с почтовыми индексами;
- **DBPS** - список подразделений предприятия;
- **POSS** - список должностей;
- **JOBS** - таблица для хранения информации о трудовой деятельности сотрудников;
- **FAMILY** - таблица для хранения информации о членах семьи сотрудника.

Согласно правилам построения баз данных, следует избегать избыточности, т.е. необходимо выделять часто повторяющиеся данные в отдельные таблицы и затем ссылаться на них по уникальным идентификаторам. Это значительно сокращает объем таблиц и упрощает выборку данных.

Например, нет смысла размещать поля о членах семьи сотрудника в таблице **STAFF**, а затем многократно дублировать персональную информацию для каждой новой строки, содержащей уникальные сведения об очередном родственнике. Правильный подход - выделить сведения о членах семьи в отдельную таблицу (**FAMILY**) и затем установить связь между ней и таблицей **STAFF** по уникальному идентификатору сотрудника.

Такой же подход следует применять и к часто изменяемым данным, наподобие названий подразделений и должностей. Например, если в таблице **STAFF** хранить непосредственно названия подразделений, то в случае каких-либо изменений в структуре предприятия на обновление данных потребуется много времени. Если же в таблице **STAFF** хранить только идентификаторы подразделений, которые ссылаются на соответствующие строки в таблице **DEPS**, то в случае переименования подразделений достаточно будет изменить только содержимое таблицы **DEPS**, что значительно проще.

- Прежде чем приступить непосредственно к созданию таблиц, следует выполнить их формальное описание и определить связи между ними.

Формальное описание таблицы REGIONS

Поле	Тип данных	Обязательное поле	Значение по умолчанию	Описание
Zip	Длинное целое	Да	—	Почтовый индекс
Area	Символьный	Нет	—	Название области, края
Region	Символьный	Нет	—	Название района
City	Символьный	Да	—	Название города, села,

				поселка и т.п.

Формальное описание таблицы DEPS

Поле	Тип данных	Обязательное поле	Значение по умолчанию	Описание
DeptiD	Короткое целое	Да	—	Уникальный идентификатор
DeptFullName	Символьный	Да	—	Полное название
DeptShortName	Символьный	Да	—	Сокращенное название
ParentDeptiD	Короткое целое	Да	0	Идентификатор «родительского» подразделения (используется в рассмотренных примерах)

Формальное описание таблицы сотрудников STAFF

Поле	Тип данных	Обязательное поле	Значение по умолчанию	Описание
ID	Длинное целое	Да	—	Уникальный идентификатор строки, на который ссылаются таблицы JOBS и FAMILY
LastName	Символьный	Да	—	Фамилия
FirstName	Символьный	Да	—	Имя
FatherName	Символьный	Да	—	Отчество
Zip	Длинное целое	Нет	—	Почтовый индекс (связь с таблицей regions)
Street	Символьный	Нет	—	Улица
House	Символьный	Нет	—	Номер дома
Tel	Символьный	Нет	—	Номер телефона
IdCode	Символьный	Да	«О»	Персональный идентификатор (например, код плательщика налогов)
TabNum	Символьный	Нет	—	Табельный номер
BirthDate	Дата	Да	—	Дата рождения
BornPlace	Символьный	Нет	—	Место рождения
DepID	Короткое целое	Да	—	Идентификатор подразделения (связь с таблицей

				dsps)
Posid	Короткое целое	Да	—	Идентификатор должности (связь с таблицей poss)
Salary	Вещественное число с двумя десятичными знаками	Да	Самый распространенный оклад	Размер оклада
PasspNum	Символьный	Нет	—	Номер паспорта
PasspDate	Дата	Нет	—	Дата выдачи паспорта
PasspOrg	Символьный	Нет	—	Организация, выдавшая паспорт
Photo	Графика	Нет	—	Фотография

Формальное описание таблицы POSS

Поле	Тип данных	Обязательное поле	Значение по умолчанию	Описание
Posid	Короткое целое	Да	—	Уникальный идентификатор
PosFullName	Символьный	Да	—	Полное название
PosShortName	Символьный	Да	—	Сокращенное название
PosLevel	Короткое целое	Да	1	Ранг должности чем меньше число, тем выше уровень (используется в рассмотренных примерах)

Формальное описание таблицы JOBS

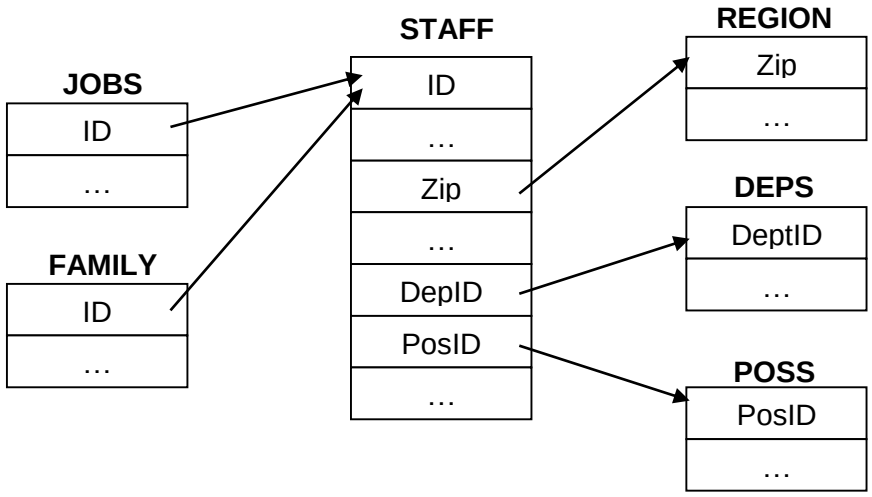
Поле	Тип данных	Обязательное поле	Описание
ID	Длинное целое	Да	Идентификатор строки в таблице STAFF
StartDate	Дата	Да	Дата зачисления на должность
StopDate	Дата	Нет	Дата освобождения с должности
Organization	Символьный	Да	Название организации
Dept	Символьный	Нет	Название подразделения
Pos	Символьный	Да	Название должности
StopCause	Символьный	Нет	Причины освобождения с занимаемой должности
Curorg	Символьный	Нет	Поле содержит 1. если строка относится к периоду работы сотрудника в текущей организации. В противном случае поле содержит 0.

InOrderNum	Символьный	Нет	Номер приказа о зачислении на должность
OutOrdertNum	Символьный	Нет	Номер приказа об освобождении с занимаемой должности

Формальное описание таблицы FAMILY

Поле	Тип данных	Обязательное поле	Описание
ID	Длинное целое	Да	Идентификатор строки в таблице staff
Kin	Символьный	Да	Степень родства (сын, дочь, муж, жена, брат, сестра, мать, отец)
BirthDate	Дата	Нет	Дата рождения члена семьи
KinName	Символьный	Да	Имя члена семьи

- Связи между таблицами представлены на рисунке.



- После составления формального описания таблиц и определения связей между ними можно приступать к их непосредственному созданию.
- Для создания таблиц предназначен SQL-оператор **create table**. Его синтаксис в простейшем случае имеет следующий вид:

CREATE TABLE имя_таблицы
(имя_поля! тип_данных атрибуты,
имя_поля2 тип_данных атрибуты,
…
имя_поляN тип_данных атрибуты)

- Имя таблицы и имена полей в кавычки не заключаются. Необязательные атрибуты могут состоять из одного или нескольких элементов, отделяемых друг от друга пробелом. Согласно стандарту SQL2 допустимы следующие атрибуты полей:
    - **DEFAULT** значение_по_умолчанию — определяет значение автоматически подставляемое в столбец при создании новой строки;
    - **NOT NULL** — указывает на то, что поле обязательно должно содержать какие-либо данные;
    - **UNIQUE** — указывает на то, что столбец не может содержать повторяющихся значений в различных строках.
- Таким образом, SQL-оператор создания таблицы **STAFF** будет выглядеть следующим образом:

```
CREATE TABLE "STAFF"
(
 "ID"       INTEGER NOT NULL,
 "LastName"     VARCHAR(30) NOT NULL,
 "FirstName"    VARCHAR(30) NOT NULL,
 "FatherName" VARCHAR(30),
 "Zip"      INTEGER,
 "Street" VARCHAR(30),
 "House" VARCHAR(10),
 "Tel"      VARCHAR(20),
 "IdCode"       VARCHAR(20) DEFAULT '0' NOT NULL,
 "TabNum"       VARCHAR(20),
 "BirthDate"    DATE NOT NULL,
 "BornPlace"    VARCHAR(50),
 "DepID"        SMALLINT NOT NULL,
 "PosID" SMALLINT NOT NULL,
 "Salary"       NUMERIC(15, 2) DEFAULT 10000.00 NOT NULL,
 "PasspNum"     VARCHAR(20),
 "PasspUate"    DATE,
 "PasspOrg"     VARCHAR(50),
 "PhuLu"        BLOB SUB_TYPE 0 SEGMENT SIZE 80,
```

UNIQUE ("ID"),

UNIQUE ("IdCode")

);

- После выполнения этого SQL-оператора появится соответствующий элемент в категории **STAFF.GDB - Tables** программы **IBConsole**.
- Аналогично создайте остальные таблицы базы данных самостоятельно.

Задание 2. Оператор ALTER TABLE

Методические указания по выполнению задания:

1. Оператор **ALTER TABLE** используется для добавления и удаления столбцов и значений по умолчанию в существующей таблице, а также для изменения набора ограничении.
2. Для того чтобы добавить столбец в таблицу, используется следующая запись:

   ALTER TABLE имя_таблицы

   ADD имя_столбца тип_данных атрибуты
3. Например, можно добавить еще один столбец в таблицу **STAFF**:

   ALTER TABLE "STAFF"

   ADD "NewColumn" INT NOT NULL
4. Для удаления из таблицы поля или элемента таблицы, используют оператор **ALTER TABLE** следующего вида:

   ALTER TABLE имя_таблицы

   DROP имя_столбца
5. Пример - удаление столбца **NewColumn** таблицы **STAFF**:

   ALTER TABLE "STAFF"

   DROP "NewColumn"
6. С помощью оператора **ALTER TABLE** можно добавить к существующей таблице ограничение типа **UNIQUE**, рассмотренное ранее. Для этого используется запись вида:

   ALTER TABLE имя_таблицы

   ADD CONSTRAINT имя_ограничения UNIQUE (столбцы)
7. Конструкция **CONSTRAINT имя_ограничения** необязательна, однако ее лучше использовать, поскольку в противном случае созданное ограничение впоследствии будет невозможно удалить средствами языка SQL.
8. Кроме рассмотренных ранее ограничений, определяющих столбцы с обязательными и

уникальными значениями, существуют еще три часто используемых типа ограничений:

- проверка значений;
- первичный ключ;
- вторичный ключ.

9. Проверка значений столбца устанавливается с помощью ключевого слова **CHECK**, после которого указывают требуемое условие. Для добавления такого ограничения используют запись вида:

```
ALTER TABLE имя_таблицы
    ADD CONSTRAINT имя_ограничения CHECK (условие)
```

10. Имя ограничения указывать не обязательно, однако рекомендуется, поскольку по этому имени пользователю будет проще определить источник проблемы при выдаче сообщения о нарушении установленного ограничения. Кроме того, в случае необходимости при наличии имени ограничение можно впоследствии удалить.
11. При составлении условия проверки могут использоваться ключевые слова **AND, OR** и **IN**, операции сравнения =, <, >, <= и >=, а также конструкция **BETWEEN ... AND**.
12. Создадим проверку для столбца **Zip** в таблицах **STAFF** и **REGIONS**, чтобы вводимые значения не выходили за пределы диапазона 1...999999:

```
ALTER TABLE "STAFF"
    ADD CONSTRAINT "INVALID_ZIP_STAFF"
    CHECK ("Zip" BETWEEN 1 AND 999999);
ALTER TABLE "REGIONS"
    ADD CONSTRAINT "INVALID_ZIP_REGIONS"
    CHECK ("Zip" BETWEEN 1 AND 999999);
```

13. Пример - ограничение на значение оклада сотрудника (должен быть больше или равен нулю):

```
ALTER TABLE "STAFF"
    ADD CONSTRAINT "INVALID_SALARY" CHECK ("Salary" >= 0);
```

14. Ограничение для таблицы **JOBS**. Его смысл заключается в том, что индикатор строк, соответствующий должностям в текущей организации, должен содержать только 1 или 0.

```
ALTER TABLE "JOBS"
    ADD CONSTRAINT "INVALID_CURORG" CHECK ("CurOrg" IN ( '1','0'));
```

15. **Первичный ключ** — это столбец (или набор столбцов), который однозначно

идентифицирует каждую строку таблицы. При этом столбцы, входящие в состав первичного ключа, обязательно должны быть расположены первыми в списке столбцов. Для добавления первичного ключа к уже существующей таблице используется запись вида:

```
ALTER TABLE имя_табпицы
    ADD CONSTRAINT имя_первичного_ключа
    PRIMARY KEY (столбцы)
```

16. Конструкции **CONSTRAINT имя_первичного_ключа** необязательна, однако ее лучше использовать, поскольку в противном случае созданный первичный ключ впоследствии будет невозможно удалить средствами языка SQL.
17. Например, для таблицы STAFF это выглядит следующим образом:

```
ALTER TABLE "STAFF"
    ADD CONSTRAINT "PK_STAFF" PRIMARY KEY ("ID");
```

18. Для того чтобы определить первичный ключ при создании таблицы, можно в описании поля указать атрибут **PRIMARY KEY**. Для таблицы **STAFF** -это бы выглядело следующим образом:

```
CREATE TABLE "STAFF"
    ("ID" INTEGER NOT NULL PRIMARY KEY,
    …
    UNIQUE ("IdCode") )
```

19. Действие атрибутов **UNIQUE** и **PRIMARY KEY** одинаково, в связи с чем, для столбца, входящего в первичный ключ, атрибут **UNIQUE** можно опустить. Именно поэтому в разделе unique представленного выше оператора указан только столбец **IDCode**.
20. Однако такой способ определения первичного ключа поддерживается не всеми диалектами языка SQL. Стандартным считается способ определения первичного ключа в особом разделе в конце оператора **CREATE TABLE**:

```
CREATE TABLE "STAFF"
    ("ID" INTEGER NOT NULL,
    …
    UNIQUE ("IdCode"),
    PRIMARY KEY ("ID"));
```

21. Однако у такого способа есть один недостаток: первичный ключ будет невозможно

удалить средствами языка SQL, если возникнет такая необходимость.

22. Аналогично определяется первичный ключ для таблиц **REGIONS, DEPS** и **POSS**:

```
ALTER TABLE "REGIONS"
    ADD CONSTRAINT "PK_REGIONS" PRIMARY KEY ("Zip");
ALTER TABLE "DEPS"
    ADD CONSTRAINT "PK_DEPS" PRIMARY KEY ("DeptID");
ALTER TABLE "POSS"
    ADD CONSTRAINT "PK_POSS" PRIMARY KEY ("PosID");
```

23. **Внешний ключ** определяет столбец одной таблицы, значения которого должны существовать во второй таблице, называемой внешней. В отличие от первичного ключа, внешний ключ не является уникальным индексом строки. Уникальными являются столбцы во внешней таблице, которые обязательно должны входить в первичный ключ.
24. Так же, как и первичный, внешний ключ можно определять в операторе **CREATE TABLE** или после создания таблицы с помощью оператора alter table. В последнем случае используется запись вида:

```
ALTER TABLE имя_таблицы
    ADD CONSTRAINT имя_внешнего_ключа
            FOREIGN KEY (столбцы)
            REFERENCES внешняя_таблица (ключевые_столбцы)
```

25. Типы данных для полей, по которым создается внешний ключ, должны совпадать с типами данных соответствующих ключевых полей во внешней таблице.
26. Конструкция **CONSTRAINT имя_внешнего_ключа** необязательна, однако ее лучше использовать, поскольку в противном случае созданный вторичный ключ впоследствии будет невозможно удалить средствами языка SQL.
27. Внешний ключ для таблицы **STAFF**:

```
ALTER TABLE "STAFF"
    ADD CONSTRAINT "FK_STAFF_POSID" FOREIGN KEY ("POSID")
    REFERENCES "POSS";
```

28. Если названия ключевых столбцов в главной и внешней таблицах совпадают, то для внешней таблицы их в операторе **ALTER TABLE** можно не указывать. Данное ограничение назначает столбец **POSID** внешним ключом, который ссылается на аналогичный столбец таблицы **POSS**. Это означает, что введенные в таблице **STAFF**

идентификаторы должностей должны существовать в таблице **POSS**. Кроме того, идентификаторы, используемые в таблице **STAFF**, нельзя удалить из таблицы **POSS**. Такая возможность закрепления связи между двумя таблицами называется ссылочной целостностью.

29. Аналогичный внешний ключ для поля **DepID**:

```
ALTER TABLE "STAFF"
    ADD CONSTRAINT "FK_STAFF_DEPID" FOREIGN KEY ("DepID")
    REFERENCES "DEPS" ("DeptID")
```

30. В приведенном примере названия полей в таблицах **STAFF** и **DEPS** не совпадают, поэтому после названия таблицы **DEPS** указывается имя столбца, с которым связан внешний ключ таблицы **STAFF**. Определим для таблицы **STAFF** еще один внешний ключ — по почтовому индексу:

```
ALTER TABLE "STAFF"
    ADD CONSTRAINT "FK_STAFF_ZIP" FOREIGN KEY ("Zip")
    REFERENCES "REGIONS"
```

31. Для удаления ограничений, которым было назначено имя, используется оператор **ALTER TABLE** следующего вида:

```
ALTER TABLE имя_таблицы
    DROP CONSTRAINT имя_ограничения
```

32. Пример - удаление внешнего ключа fk_staff_zip из таблицы STAFF:

```
ALTER TABLE "STAFF"
    DROP CONSTRAINT "FK_STAFF_ZIP"
```

33. Для удаления из базы данных ненужной таблицы используется оператор следующего вида:

```
DROP TABLE имя_таблицы
```

34. При этом для удаления таблицы, с которой связаны какие-либо объекты базы данных, следует использовать ключевые слова restrict или cascade:

```
DROP TABLE имя_таблицы RESTRICT
DROP TABLE имя_таблицы CASCADE
```

35. Ключевое слово **RESTRICT** при наличии в базе данных объектов, зависимых от удаляемой таблицы, вызовет отмену операции удаления. В этой же ситуации ключевое слово **CASCADE** приведет к удалению всех зависимых объектов.

Внеаудиторная самостоятельная работа:

Составить опорный конспект письменно в тетради по основным командам создания, модификации и удаления таблиц.

Практическая работа №16

Работа с индексами. Создание, модификация и удаление индексов

Цель работы: сформировать умения по выполнению основных команд по созданию, модификации и удалению индексов в **InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**.

Теоретическая справка:

Индекс — это механизм физического хранения данных, позволяющий ускорить операции поиска строк в таблице. Использование индексов дает возможность отказаться от последовательного просмотра всех строк таблицы - данные в индексе сортируются таким образом, чтобы обеспечить максимально быстрый поиск.

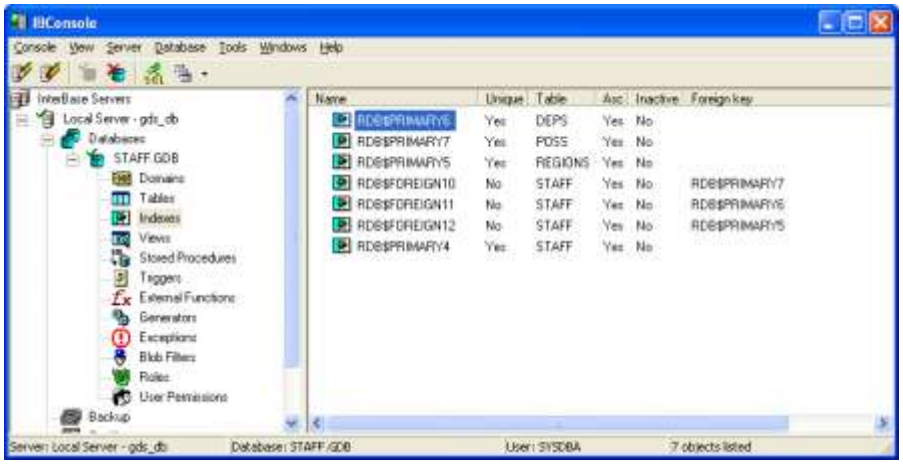
Хотя индексы не поддерживаются стандартом SQL, возможность их построения

реализована во всех современных системах управления базами данных - как локальных, так и клиент-серверных. В случае с большими таблицами правильно организованные индексы позволяют значительно ускорить процесс выборки данных — особенно в клиент-серверных системах, где реализована автоматическая оптимизация SQL-запросов на основании информации об индексированных полях.

С физической точки зрения в различных базах данных индексы реализованы по-разному Например, в таблицах dBASE и FoxPro расширения индексных файлов могут отличаться в зависимости от версии: .cdx, .mdx, .idx. В тоже время для таблиц Paradox существуют понятия первичных и вторичных индексов. Первичные индексы создаются автоматически на основе первичного ключа и хранятся в отдельных файлах.

В клиент-серверных системах индекс является составным элементом базы данных и в отдельный файл не выносится. Как и в случае с таблицами Paradox, SQL-серверы автоматически создают системные индексы по полям, входящим в первичные и внешние ключи. Кроме того, автоматически создаются индексы по тем столбцам, для которых установлен атрибут **UNIQUE**. Это наглядно прослеживается в базах данных **InterBase**.

Так, в соответствии с определением столбцов с уникальными значениями в таблицах **STAFF**, **REGIONS**, **DEPS** и **POSS**, SQL-сервер **InterBase** автоматически создал индексы, показанные на рисунке.



Задание 1. Создание индексов. Оператор CREATE INDEX

Методические указания по выполнению задания:

1. Для создания пользовательских индексов в расширениях языка SQL предусмотрена команда **CREATE INDEX**, базовый синтаксис которой в различных диалектах идентичен:

CREATE INDEX имя_индекса

ON имя_таблицы (поле1, поле2, ..., полеN)

2. По умолчанию все столбцы, входящие в индекс, сортируются по возрастанию. Тем не менее для каждого из них можно указать порядок сортировки явно, при создании индекса. Для этого используются ключевые слова **ASC** и **DESC**. Например, если для таблицы **TABLE1** необходимо создать индекс **Index1** по полям **FIELD1** (сортировка в порядке убывания) и **FIELD2** (сортировка в порядке возрастания), то соответствующие SQL-оператор будет выглядеть следующим образом:

   CREATE INDEX INDEX1 ON TABLE1 (FIELD1 DESC, FIELD2 ASC)

3. Кроме того, команда **CREATE INDEX** поддерживает создание индексов с уникальностью значений и индексируемых столбцах. Для этого используется ключевое слово **UNIQUE**, например:

   CREATE UNIQUE INDEX INDEX2 ON TABLE1 (FIELD3)

   Это равнозначно явному указанию атрибута **UNIQUE** для поля **FIELD3** при создании таблицы **TABLE1**. В результате уникальность значений в столбце будет автоматически отслеживаться системой управления базой данных.

4. Создадим несколько индексов для нашего примера базы данных **STAFF**. Прежде всего, проиндексируем таблицу **STAFF** по фамилиям, именам и отчествам сотрудников:

   CREATE INDEX STAFF_INDEX1

   ON "STAFF" ("LastName", "FirstName", "FatherName");

5. Для этой же таблицы создадим еще два уникальных индекса: по столбцу с идентификационным кодом и по столбцу с табельным номером:

   CREATE UNIQUE INDEX STAFF_INDEX2

   ON "STAFF" ("IdCode");

   CREATE UNIQUE INDEX STAFF_INDEX3

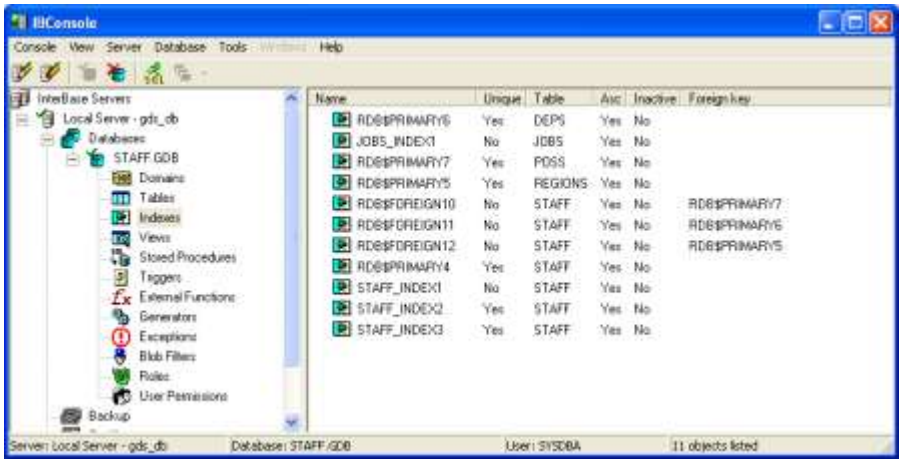
   ON "STAFF" ("TabNum");

6. Создадим также индекс для таблицы **JOBS**, который будет определять сортировку строк по сотрудникам и датам вступления на должность:

   CREATE INDEX JOBS_INDEX1

   ON "JOBS" ("ID", "StartDate");

7. Теперь, в случае SQL-сервера **InterBase**, созданные индексы отобразятся в программе **IBConsole** следующим образом:



Обратите внимание на поле подсказки, в котором указано: «Индекс ускоряет поиск и сортировку в данном поле, но замедляет обновление». Речь идет о том, что при каждом изменении данных в индексируемых столбцах, соответствующие индексы автоматически обновляются (перестраиваются). Таким образом, чем больше в базе данных индексов (и чем больше в них включено данных), тем больше времени требуется на их обновление.

Учитывая это, создавать слишком много индексов нежелательно, поскольку вместо повышения производительности они могут привести к обратному результату — замедлению выборки данных. В частности, не имеет смысла создавать индексы для таблиц, содержащих несколько десятков строк. Кроме того, рекомендуется создавать индексы только для тех столбцов, которые часто используются при выборке данных SQL-командой **SELECT**.

Следует также быть внимательным при работе с очень большими таблицами. Для таких таблиц вместо первичных ключей лучше использовать уникальные индексы, поскольку при попытке продублировать значение некоторого уникального поля появится сообщение об исключительной ситуации, когда вместо закодированного будет использоваться нормальное имя индекса. Кроме того, многие системы управления базами данных поддерживают временное отключение и перестраивание уникальных индексов, чего нельзя сказать о первичных ключах.

8. Временное отключение индексов поддерживается, в частности, и при работе с SQL-сервером **InterBase**. Для этого служит SQL-команда:

   ALTER INDEX имя_ индекса INACTIVE.

9. Для повторного подключения и перестраивания индекса используется SQL-команда:

   ALTER INDEX имя_индекса ACTIVE.

10. С внешними ключами дело обстоит несколько иначе. Для поля **Zip** в таблице **STAFF**

значения выбираются строго из таблицы **REGIONS**. С этой целью создается внешний ключ, при объявлении которого создастся индекс по полю **Zip**. Проблема заключается в том, что этот индекс может значительно понизить производительность. Если предположить, что таблица **STAFF** состоит из сотен тысяч записей (хотя, конечно же, такое предположение для базы данных «Персонал» является чисто теоретическим) и в ней встречаются ссылки всего на десять регионов, то на каждый регион в индексе но полю Zip будет приходится число записей, измеряемое десятками тысяч. Такие индексы использовать крайне нежелательно, поскольку они могут вызвать проблемы с оптимизацией SQL-запросов.

11. Кроме того, следует учитывать, что при индексации по нескольким столбцам SQL-сервер может оптимизировать поиск информации только в тех полях, которые указаны в индексе первыми. Например, в базе данных **STAFF** есть индекс **STAFF_INDEX1** по полям **LastName, FirstName**, **FatherName** таблицы **STAFF**. Если организовать выборку данных из поля **FatherName,** то этот индекс в оптимизации запроса использован не будет. Таким образом, с целью оптимизации выборки данных лучше использовать индексы по отдельным полям. И еще:
    - не рекомендуется создавать два индекса, начинающихся с одного и того же поля, поскольку это мешает оптимизации запросов;
    - не следует создавать много индексов по нескольким столбцам;
    - для небольших таблиц лучше ограничиться первичным ключом или уникальным индексом (либо вообще не создавать никаких индексов).

Задание 2. Удаление индексов. Оператор DROP INDEX

Методические указания по выполнению задания:

1. Для удаления индекса из базы данных используется SQL-оператор **DROP INDEX**. При этом для разных SQL-серверов его синтаксис может несколько отличаться. Для **InterBase** синтаксис этой команды выглядит:

   DROP INDEX имя_индекса
2. Системные индексы, создаваемые автоматически для полей с атрибутом **UNQUE**, а также для первичных и вторичных ключей, удалить с помощью команды **DROP INDEX** невозможно.

Внеаудиторная самостоятельная работа:

Составить опорный конспект письменно в тетради по основным командам создания, модификации и удаления индексов в **InterBase**.

Практическая работа №17

Работа с представлениями. Создание, модификация и удаление представлений

Цель работы: сформировать умения по выполнению команды по созданию, модификации и удалению представлений в **InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**.

Теоретическая справка:

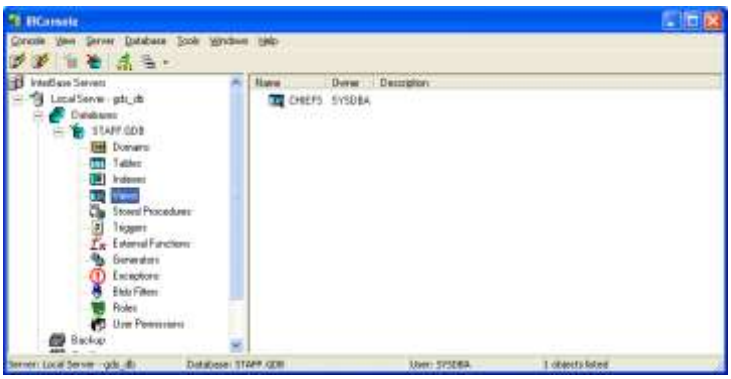
1. **Представление таблицы SQL** создается с помощью оператора **SELECT**, после чего с полученным набором данных можно обращаться как с таблицей и применять к нему операторы **SELECT** (который затем можно применять к полученному набору данных, имеющему вид таблицы). В самом представлении не хранится никаких данных — это всего лишь маленькая SQL-программа.
2. Хотя стандартом SQL представления не поддерживаются, операторы для работы с ними

реализованы во всех современных клиент-серверных системах, включая **Microsoft Office Access** и **Borland InterBase**.

Задание 1. Создание представлений. Оператор CREATE VIEW

Методические указания по выполнению задания:

1. Представления создаются с помощью SQL-оператора **CREATE VIEW** следующего вида:
   CREATE VIEW имя_представления(имена_столбцов) AS
   SELECT ...
2. Перечень имен столбцов, входящих в состав представления, можно опустить:
   CREATE VIEW имя__представления AS
   SELECT . . .
   В этом случае имена результирующих столбцов будут совпадать с именами столбцов в таблицах, указанных после ключевого слова **SELECT**.
3. Правила построения оператора **SELECT**, используемого для создания представления, аналогичны тем правилам, которые применяют при работе с обычными операторами **SELECT**. Единственное исключение состоит в том, что внутри оператора **CREATE VIEW** не может использоваться операция сортировки **ORDER BY**.
4. Рассмотрим пример запроса, выбирающего фамилии, имена и отчества всех сотрудников на руководящих должностях. На основании данного запроса можно построить следующее представление:
   CREATE VIEW CHIEFS_1 AS
   SELECT STAFF."LastName", STAFF."FirstName", STAFF."FatherName", POSS."PosFullName"
   FROM STAFF, POSS
   WHERE (STAFF."PosID" = POSS."PosID") AND (STAFF."PosID" IN (SELECT "PosID" FROM POSS WHERE "PosLevel" BETWEEN 1 AND 11));
5. В среде **InterBase** для доступа к представлениям используется элемент **Views** в иерархической структуре базы данных.



6. Во всем остальном работа с представлениями в программе **IBConsole** ничем не отличается от работы с таблицами.

Задание 2. Использование псевдонимов столбцов

Методические указания по выполнению задания:

1. Рассмотрим случай, когда при создании представления **CHIEFS** для результирующих столбцов определен новый набор имен:

   CREATE VIEW CHIEFS_1 ("Фамилия", "Имя", "Отчество", "Должность") AS
   SELECT STAFF."LastName", STAFF."FirstName", STAFF."FatherName", POSS."PosFullName"
   FROM STAFF, POSS
   WHERE (STAFF."PosID" = POSS."PosID") AND (STAFF."PosID" IN (SELECT "PosID" FROM POSS WHERE "PosLevel" BETWEEN 1 AND 3))

2. Теперь из представления **CHIEFS_1** можно выбирать данные с помощью оператора **SELECT** как из обычной таблицы. Например, выберем все данные из представления **CHIEFS_1**, отсортировав их вначале по должностям, а затем — по фамилиям, именам и отчествам.
3. Запрос в формате **Borland InterBase** имеет следующий вид:

   SELECT "Должность", "Фамилия", "Имя", "Отчество"
   FROM CHIEFS_1
   ORDER BY "Должность", "Фамилия", "Имя", "Отчество"

4. При выборке данных из представлений необходимо учитывать особенности синтаксиса конкретного SQL-сервера — и прежде всего при использовании агрегатных функций и вложенных запросов. Например, если мы хотим подсчитать количество однофамильцев среди руководителей, то для **InterBase** должен быть использован запрос следующего вида

(в Microsoft Office Access 2003 неприменим):

SELECT COUNT(DISTINCT "Фамилия") FROM CHIEFS_1

Задание 3. Использование объединений

Методические указания по выполнению задания:

1. При создании представлений часто используют объединения. Например, в таблице **STAFF** указаны почтовые индексы и идентификаторы подразделений и должностей, а соответствующие данные находятся в таблицах **REGIONS**, **DEPS** и **POSS**. Создадим представление **FULLLIST**, формирующее единый набор данных о сотрудниках:

   CREATE VIEW FULLLIST AS
   SELECT STAFF."ID", STAFF."LastName", STAFF."FirstName",
   STAFF."FatherName", STAFF."Zip", REGIONS."Area", REGIONS."Region",
   REGIONS."City", STAFF."Street", STAFF."House", STAFF."Tel", STAFF."IdCode",
   STAFF."TabNum", STAFF."BirthDate", STAFF."DepID", DEPS."DeptShortName",
   DEPS."ParentDeptiD", STAFF."PosID", POSS."PosShortName", POSS."PosLevel",
   STAFF."Salary"
   FROM ((STAFF INNER JOIN REGIONS ON STAFF."Zip" = REGIONS."Zip")
   INNER JOIN DEPS ON STAFF."DepID" = DEPS."DeptID") INNER JOIN POSS ON
   STAFF."PosID" = POSS."PosID"

2. Это представление выбирает информацию не только из таблицы **STAFF**, но и соответствующие данные по каждому сотруднику из таблиц **REGIONS, DEPS** и **POSS**. Теперь в каждой строке представления **FULLLIST** будет не только идентификатор подразделения, но и его сокращенное название и идентификатор «родительского» подразделения. С такими представлениями во многих случаях работать проще, чем непосредственно с таблицами, поскольку отпадает необходимость постоянно подавать вложенные запросы и объединения. Например, для получения отсортированного списка сотрудников на руководящих должностях теперь можно воспользоваться следующим запросом:

   SELECT "PosShortName", "LastName", "FirstName", "FatherName"
   FROM FULLLIST WHERE "PosLevel" < 4
   ORDER BY "PosShortName", "LastName", "FirstName", "FatherName"

3. Тем не менее в каждом конкретном случае необходимо рассматривать, какой из вариантов

получения результирующего набора данных наиболее эффективен. Так, при выборке сведений о руководителях с помощью представления **CHIEFS_1** вначале формируется небольшой набор данных о должностях, а затем на его основе извлекаются строки из самой большой таблицы **STAFF**. Если же воспользоваться представлением **FULLLIST**, то вначале будет сформирован массивный набор данных о сотрудниках, из которого затем выбираются строки с **PosLevel < 4**. Очевидно, что в том случае, когда таблица **STAFF** состоит из тысяч строк, последний способ менее эффективен.

Задание 4. Удаление представлений. Оператор DROP VIEW

Методические указания по выполнению задания:

1. В отличие от таблиц оператор **ALTER** для представлений не используется. Таким образом, если необходимо внести изменения в структуру некоторого представления, его необходимо вначале удалить, а затем создать заново. Для удаления представлений используется SQL-оператор **DROP VIEW**, например:

```
DROP VIEW CHIEFS
```

Внеаудиторная самостоятельная работа:

Составить опорный конспект письменно в тетради по основным командам создания, модификации и удаления представлений.

Практическая работа №18

Разработка клиентской части приложения. Размещение визуальных и не визуальных компонентов. Соединение с базой данных

Цель работы: сформировать умения по работе с основными компонентами **Delphi** для разработки клиентской части приложения для работы с базами данных **InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**, интегрированная среда разработчика **Delphi.**

Задание 1. Создание базы данных в InterBase

Выполнить проектирование базы данных для продажи товаров со склада различным организациям.

Пусть проектируемая база данных будет определена структурой следующих таблиц:

- **Таблица «Товар» (Tovar)**

Имя атрибута	**Описание**	**Тип данных**
TovarID	Поле-счетчик	Счетчик
Name	Название товара	Текст(100)
Price	Цена товара	Вещественный

- **Таблица «Фирмы» (Firm)**

Имя атрибута	Описание	Тип данных
FirmID	Поле-счетчик	Счетчик
Name	Название организации	Текст(100)

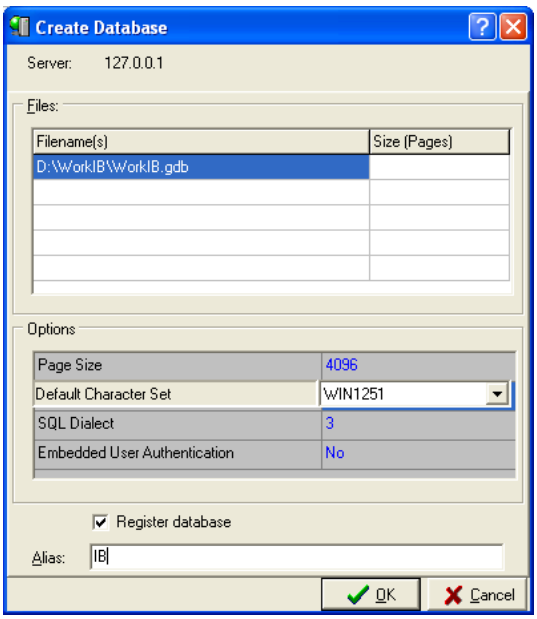
- **Таблица «Продажи» (Sale)**

Имя атрибута	Описание	Тип данных
SaleID	Поле-счетчик	Счетчик
FirmKod	Код фирмы, совершающей покупку	Число
TovarKod	Код товара, который купили	Число
Rem	Комментарий по покупке	Текст (100)

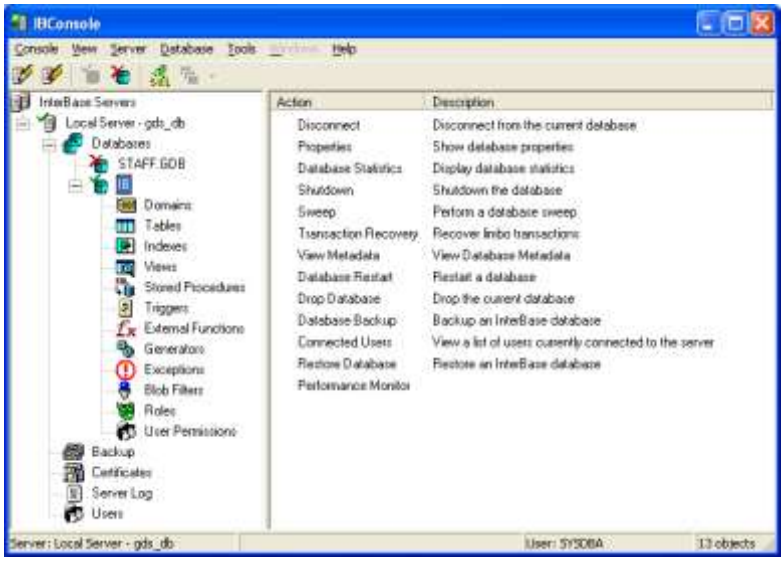
Методические указания по выполнению задания:

Создание базы данных

1. Запустите сервер **InterBase**.
2. Создайте рабочую папку для будущего проекта **D:\WorkIB.** Для каждого проекта необходимо создавать отдельную папку с осмысленным названием.
3. Создаем базу данных. Выбираем пункт меню **Database - Create Database.** Появляется окно **Create Database**. Указываем название файла базы данных и его месторасположение (**D:\WorkIB\WorkIB.gdb**) в первой строке списка **Files**. Расширение для баз данных **InterBase — GDB**. В выпадающем списке **Default Character Set** выбираем пункт **WIN1251**. Это необходимо для того, чтобы нормально отображались русские буквы.
4. В поле **Alias** вводим название будущей базы данных (**IB**). Остальное оставляем по умолчанию.



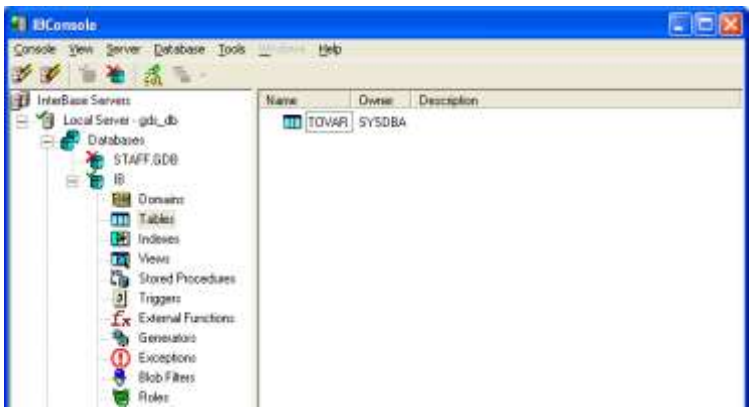
5. Нажимаем **ОК** - база создана, окно **IBConsole** приобретает следующий вид:



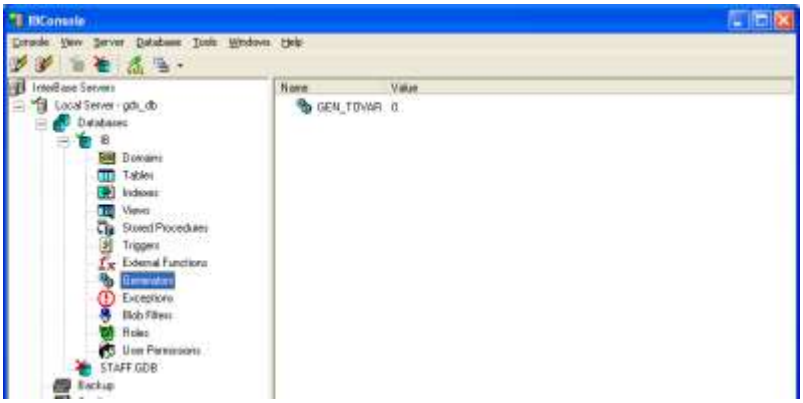
6. Теперь переходим к созданию таблиц. Выбираем в дереве в левой половине окна пункт **Tables** и нажимаем кнопку **SQL** в панели инструментов. Появляется окно **Interactive SQL**. Запишем в данном окне запрос на создание таблицы «Товар»:

```
CREATE TABLE "TOVAR"
(
  "TOVARID"      INTEGER NOT NULL,
  "NAME"         VARCHAR(100) NOT NULL,
  "PRICE"        INTEGER NOT NULL,
 PRIMARY KEY ("TOVARID")
);
```

7. Выполним запрос и закроем окно **Interactive SQL,** в результате окно **IBConsol** будет иметь следующий вид:



8. В **InterBase** нет такого типа, как счетчик или **autoincrement**, выход из этой ситуации обеспечивается созданием, так называемого генератора. Генератор — это хранящаяся в базе данных программа (или скрипт), задающая при каждом обращении к ней уникальное число. Для каждого автоинкрементного атрибута в базе данных можно создать свой генератор, можно, также использовать один генератор для нескольких атрибутов. Мы пойдем по первому пути, то есть для каждого атрибута-счетчика будем создавать отдельный генератор.
9. Вызываем **Interactive SQL**. Пишем запрос присвоения генератору начального значения: **CREATE GENERATOR «GEN_TOVAR»**. Нажимаем выполнить. После этого запрос выполнится, и окно запросом очистится. Мы создали генератор **GEN_TOVAR.**
10. Теперь нам надо установить начальное значение генератора, для этого пишем запрос присвоения генератору начального значения: **SET GENERATOR «GEN_TOVAR» TO 0.** Нажимаем выполнить. При этом запрос выполнится, и окно запросов очистится.
11. Выбирая справа в дереве окна **IBConsole** пункт **Generators**, можно проверит наличие созданного генератора. А если щелкнуть два раза левой кнопкой мыши по имени генератора, то отобразится окно **Properties**, содержащей свойства выбранного генератора.



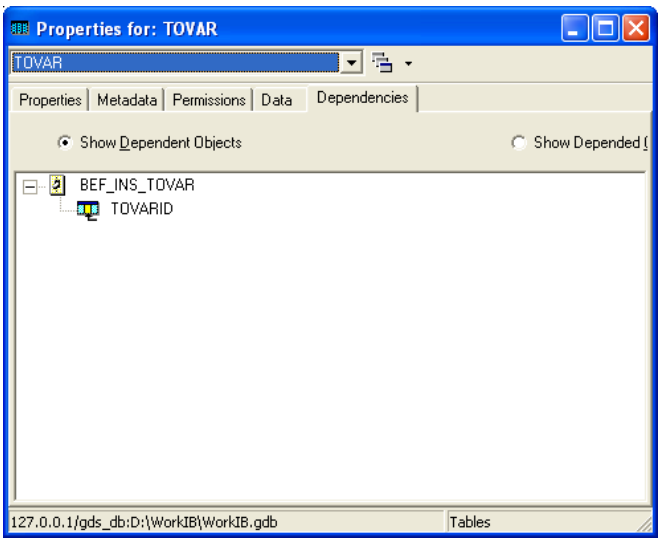
12. Теперь у нас есть генератор с заданным начальным значением. Осталось только привязать генератор к конкретному атрибуту таблицы. Пишем запрос на создание триггера:

```
CREATE TRIGGER "BEF_INS_TOVAR" FOR "TOVAR"
ACTIVE BEFORE INSERT
AS
BEGIN
NEW.TOVARID=GEN_ID(GEN_TOVAR,1);
END
```

13. Выполняется запрос, после чего генератор будет введен в работу.
14. Рассмотрим подробней то, что мы написали:

- **CREATE TRIGGER** — данная команда указывает, что мы хотим создать триггер (или правило). Далее указываем название триггера и для какой таблицы он будет предназначен.
- Предложение **ACTIVE BEFORE INSERT** — указывает, когда триггер должен выполняться, в данном случае каждый раз перед созданием новой записи.
- Слово **AS** зарезервированное, открывает тело триггера. Тело триггера всегда (даже если триггер содержит единственный оператор, как и в нашем случае) должно ограничиваться парой ключевых слов **begin** — **end.**
- В шестой строке расположен оператор, в котором новому значению (слово **new**) атрибута **tovarid** присваивается значение, полученное встроенной функции **gen_id.** Двумя параметрами обращения к это функции указывается имя генератора (**gen_tovar**) и то значение, на которое должно увеличиться текущее значение генератора («шаг» генератора), в нашем случае «шаг» равен единице.

15. Созданный триггер можно посмотреть, выбрав в окне **IBConsole** пункт **Tables** и щелкнув два раза левой кнопкой мыши на имени таблицы. Появится окно **Properties**, в данном случае оно будет отображать свойства таблицы. Необходимо выбрать вкладку **Dependencies.**



16. Нам осталось создать еще две таблицы: «**Фирмы**» и «**Продажи**». Для создания справочника организаций пишем запрос на создание таблицей «**Фирма**»:

```
CREATE TABLE "FIRM"
(
  "FIRMID"       INTEGER NOT NULL,
  "NAME"         VARCHAR(100) NOT NULL,
 PRIMARY KEY ("FIRMID")
);
```

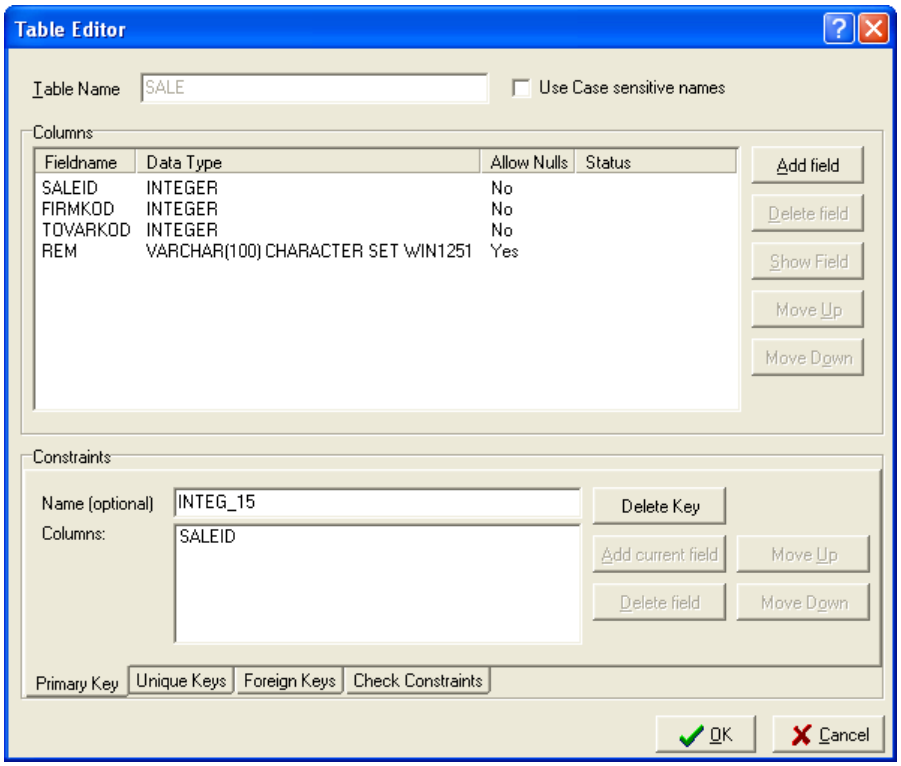
17. Создайте генератор для таблицы, написав для этого в **Interactive SQL** соответствующую конструкцию.
18. Установите начальное значение для генератора **GEN_FIRM** равным **0**.
19. Привяжите генератор **GEN_FIRM** к атрибуту **FIRM_ID** таблицы **FIRM**, создав триггер:

```
CREATE TRIGGER "BEF_INS_FIRM" FOR "FIRM"
ACTIVE BEFORE INSERT
AS
BEGIN
NEW.FIRMID=GEN_ID(GEN_FIRM,1);
END
```

20. Создадим таблицу «**Продажи**». В **InterBase** есть более простой способ создания таблиц, он заключается в том, что в левой части окна **IBConsole** выбирается пункт **Tables**, далее

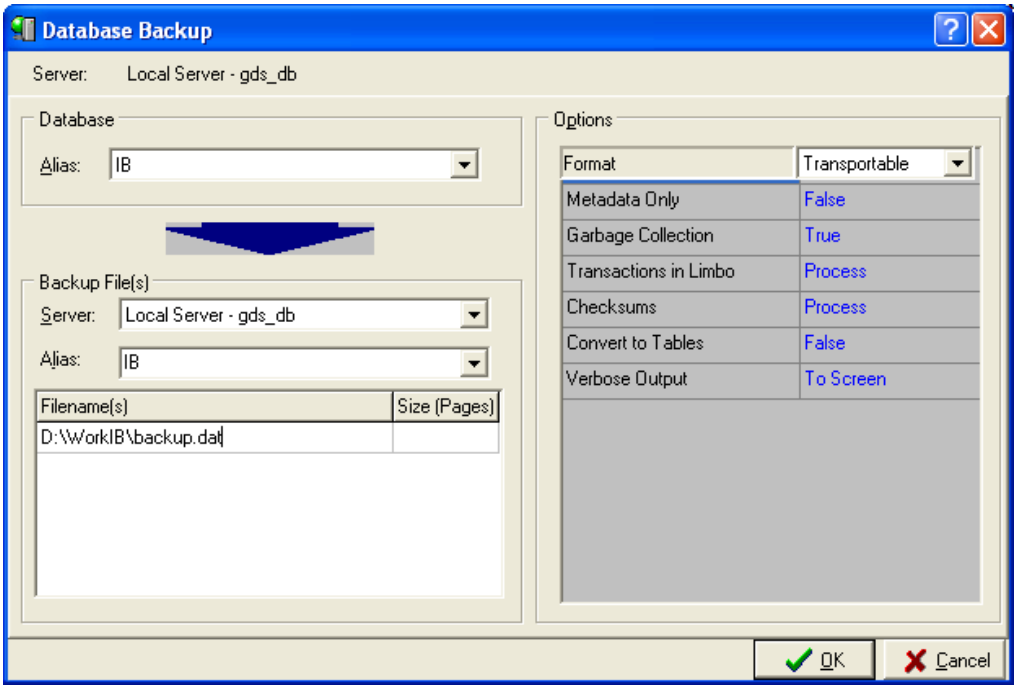
правой кнопкой мыши производится щелчок в правой части окна **IBConsole**. В выпадающем меню выбирается пункт **CREATE** (создание новой таблицы).

21. После этого появляется дополнительное окно **Table Editor**, в котором задается имя таблицы, с помощью кнопки **Add Field** добавляются атрибуты, также в этом же окне создаются первичные ключи и индексы.



22. Создайте генератор **GEN_SALE** для таблицы **SALE**. Установите его начальное значение 0. Свяжите созданный генератор и атрибут **Sale_ID**. Таким образом, нами создана база данных продажи товаров.
23. Очень важно регулярно выполнять резервное копирование базы данных. Не важно, работаете ли вы в крупной или небольшой организации, разрабатываете базу данных у себя на компьютере для диплома или курсового проекта. Никто не застрахован от случайностей и непредвиденных обстоятельств, Имея резервную копию, вы всегда можете восстановить базу данных в кратчайшие сроки, при этом до минимума сократив потраченное время и нервы, В организациях резервное копирование производится обычно каждый день. Если вы программируете у себя на компьютере, то наилучшим решением будет производить резервное копирование каждый раз, перед тем как вы соберетесь производить более или менее серьезные изменения в базе данных. Если что-то пойдет не так, то вы всегда сможете вернуть все назад.

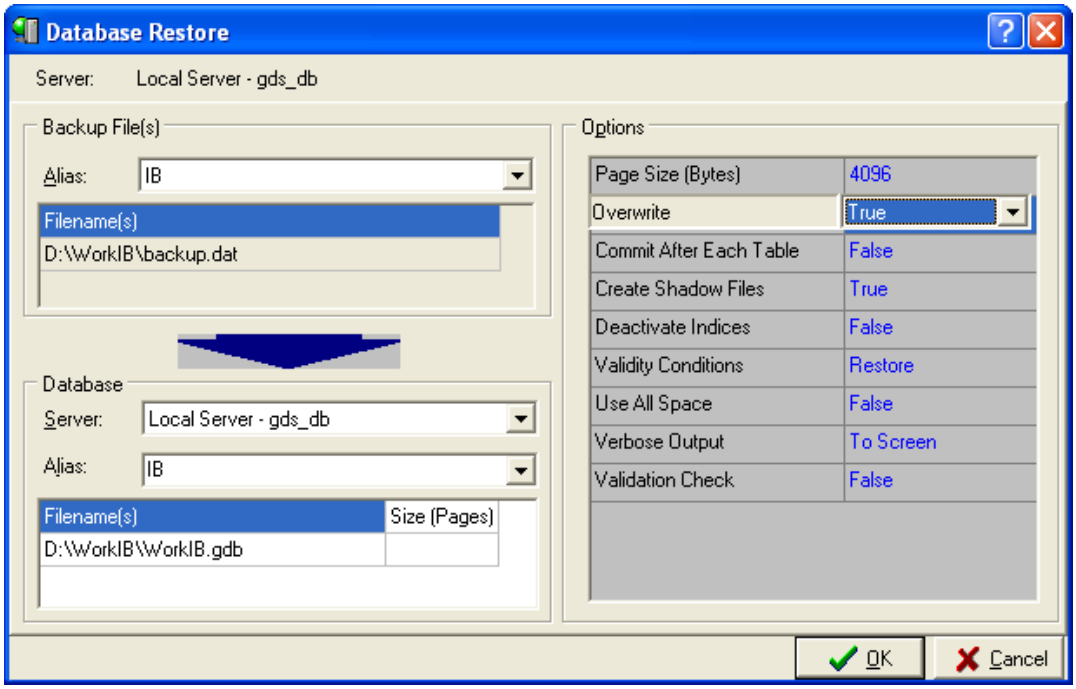
24. Для резервного копирования необходимо выбрать имя базы данных, для которой будет производиться операция (в нашем случае **IB**), щелкнуть на нем правой кнопкой мыши и в выпадающем меню выбрать пункт **Backup/Restor** затем пункт **Backup**.
25. На экране появится окно **Database Backup**. В поле **Alias** группы элементов **Database** будет указана выбранная база данных — **IB**. В группе элементов **Backup** необходимо ввести **IB** в поле **Alias** (имя базы в резервной копии) В первой строчке списка **Filename(s)** указываем **D:\WorkIB\backup.dat** — это путь и имя файла для архивной копии.



26. При нажатии на кнопку **ОК** начнется процесс резервного копирования. По его завершения появится окно **Information**, в котором сообщается, что резервное копирование завершено.
27. Нажимаем **ОК**. Закрываем окно **Database Backup**, в котором отображается информация по резервному копированию.
28. Для восстановления резервной кнопки необходимо сначала отключить соединение с данной базой. Для этого щелкаем на имени базы правой кнопкой мыши и выбираем из выпадающего меню **DISSCONECT**. На экране появится окно, в котором будет спрашиваться, уверены ли мы в том, что хотим отключить соединение с выбранной базой данных. Нажимаем **YES.**
29. Далее выбираем пункт главного меню **Database – Maitenance - Backup/Restore – Restore.** На экране появится окно **Database Restore**. В поле **Alias** группы элементов

Backup File(s) будет указана выбранная база данных — **IB**. В этой группе элементов в списке **Filename(s)** будет указано месторасположение данной копии **D:\WorkIB\backup.dat**. В группе элементов **Database** будут указаны: тип сервера — **Local Server**; имя, под которым восстановится база, — **IB**; путь, куда будет происходить восстановление, — **D:\WorkIB\WorkIB.gdb**.

30. Как видите, **InterBase** самостоятельно заполнил параметры Нам необходимо только установить в поле **Overwrite** группы **Options** значение **True**, которое будет указывать на то, что баз данных будет переписываться из резервной копии.



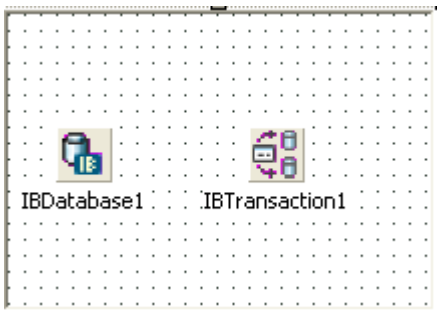
31. При нажатии **ОК** начнется процесс восстановления базы данных из резервной копии. После его завершения появится окно **Database Restore**, в котором можно будет увидеть запись **Service ended**, означающую, что операция прошла успешно. Теперь можно закрыть окно **Database Restore**.

Задание 2. Разработка приложения «клиент-сервер» в Delphi

Разработать в интегрированной среде разработчика **Delphi** приложение на основе архитектуры «клиент-сервер» для базы данных продажи товаров со склада различным фирмам.

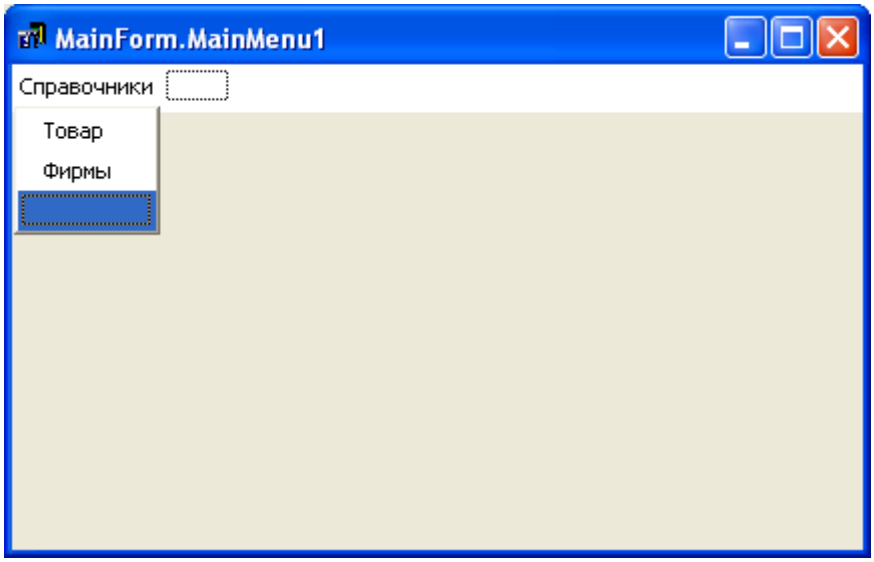
Методические указания по выполнению задания:

1. Запускаем **Delphi**. Создаем новое приложение через пункт меню **File – New - VCL Forms Application Delphi for Win32**.
2. Свойство **Name** формы меняем на **MainForm**.
3. Сохраняем проект для этого можно воспользоваться пунктом меню **File - Save All**. Выбираем рабочую папку **D:\WorkIB**. Модуль **Unit** сохраняем по именем **UMain**, а модуль приложения под именем **WorkIB.**
4. Выбираем пункт меню **Project - Options**, заходим на вкладку **Application**. В **Title** группы элементов **Application Settings** вводим «**Учет продаж**». Это будет отображаться на **Панели задач** при запуске приложения. Нажимаем **ОК**.
5. Создаем новый модуль с данными: **DataModule**. Для этого выбираем пункт меню **File – New - Other** в появившемся окне **New Items** нужно выбрать **Delphi Files** в группе элементов **Items Categories** и щелкнуть левой кнопкой мыши на пиктограмме **DataModule**, после этого нажать кнопку **ОК**.
6. В свойство **Name** пишем **DM**. Выбираем пункт меню **Files - Save All** и называем модуль **UDM**.
7. Помещаем на **DataModule** компоненты **IBDatabase**, **IBTransaction** вкладки **InterBase**. В свойство **DefaultDatabase** компонент **IBTransaction** устанавливаем значение **IBDatabase1**.

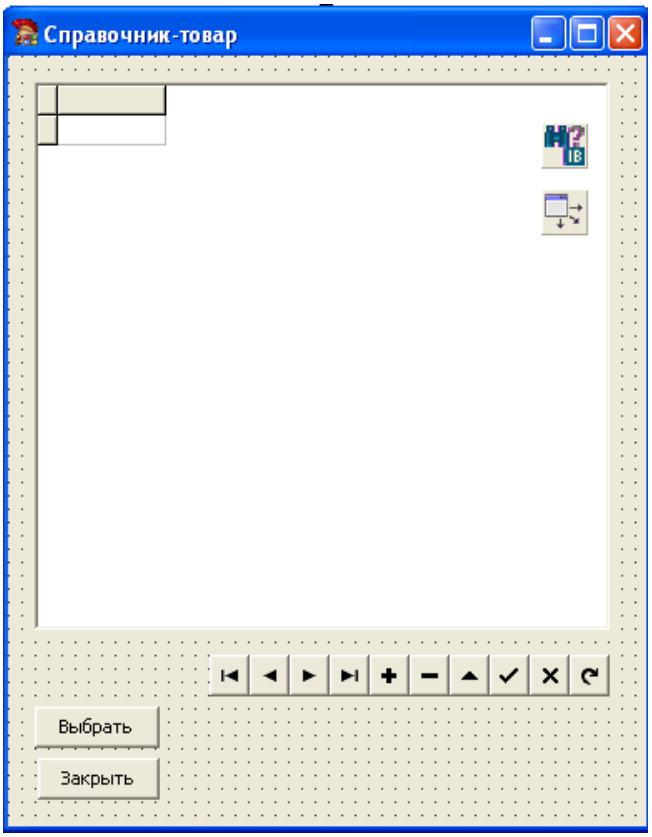


8. Для компонента **IBDatabase1** устанавливаем следующие свойства:
   - **DatabaseName** - указываем файл нашей базы **D:\WorkIB\WorkIB.gdb**;
   - в свойстве **Params** пишем: **USER_NAME=SYSDBA, PASSWORD=masterkey.** Таким образом, мы указываем, что хотим работать от имени администратора.
   - свойство **LoginPromt** ставим в **False**, что бы каждый раз при запуске программы не появлялось окошка ввода имени пользователя и его пароля.
   - свойства **Connect** устанавливаем в **True**. После этого связь с базой данных будет установлена.

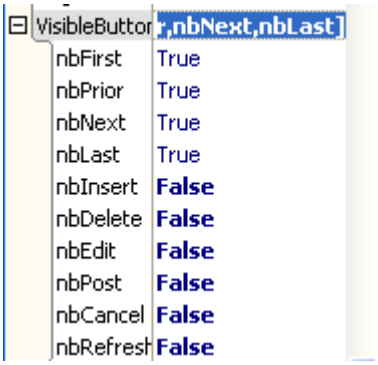
9. Помещаем на форму **MainForm** Компонент **MainMenu**, расположенный на вкладке **Standart**. Производим на нем двойной щелчок левой кнопкой мыши и создаем пункты меню как на рисунке.



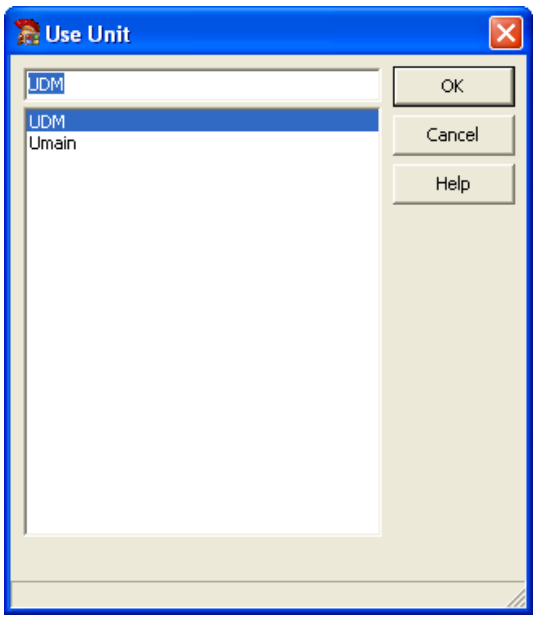
10. Создаем новую форму **File – New – Form - Delphi for Win32** называем ее **FormTovar**. В свойства **Caption** пишем «**Справочник-товар**». Сохраняем модуль под именем **UTovar**.
11. Помещаем на форму **FormTovar** компоненты **DBGrid**, **DBNavigator**, 2 компонента **Button**, **IBQuery**, **DataSource**.



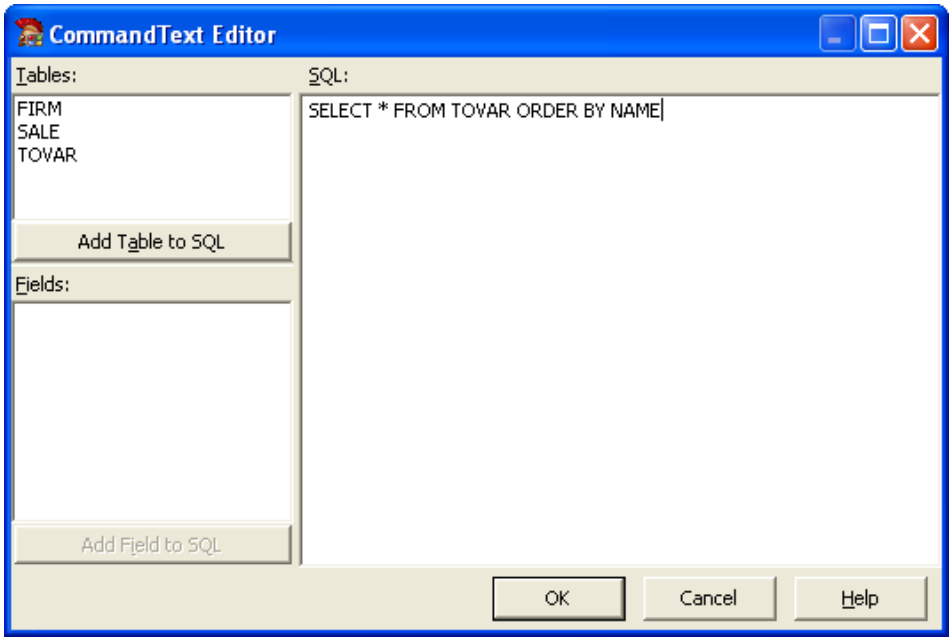
12. Свойство **Enabled** кнопки **«Выбрать»** устанавливаем в **False**, чтобы данная кнопка была доступна только в определенные моменты времени, которые мы определим сами.
13. Для компонента **DBNavigator** свойство **VisibleButtons** задаем так, как показано на рисунке.



14. В свойстве **Dataset** компонента **DataSource** устанавливаем значение **IBQuery1**. В свойстве **DataSource** компонентов **DBGrid** и **DBNavigator** ставим **DataSource1.**
15. Выбираем пункт меню **File - USE Unit**.

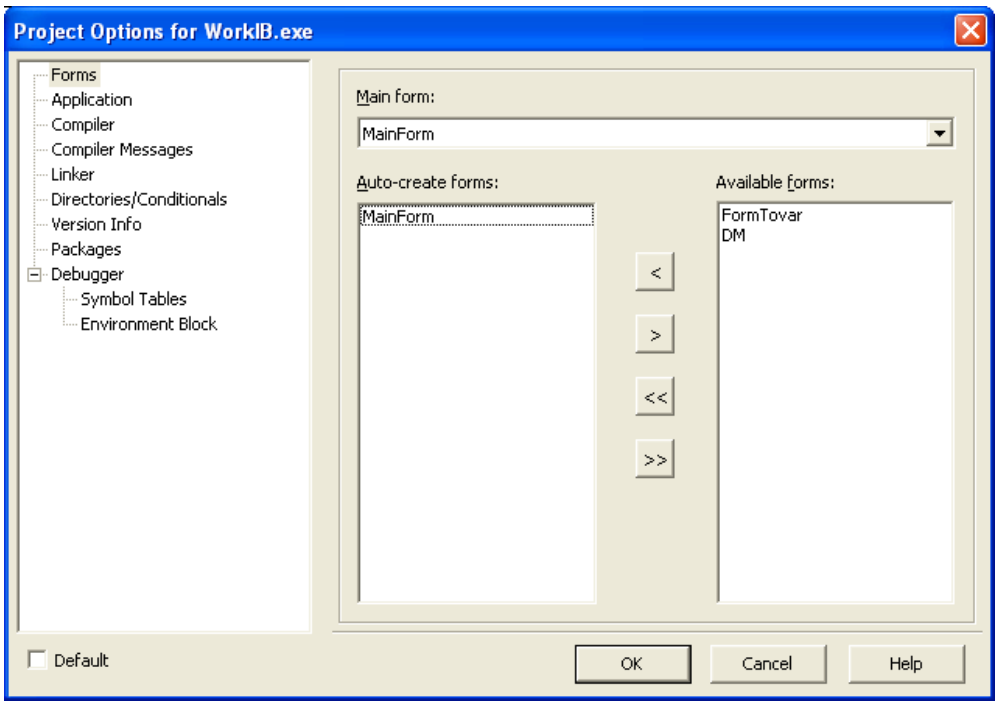


16. Выбираем **UDM** и нажимаем **ОК**. Это нужно, чтобы мы смогли подключить компонент **IBQuery1** к компоненту **IBDatabase**, так как **IBDatabase** расположен в совершенно другом модуле — **UDM**, а сейчас мы работаем с модулем **UTovar.**
17. В свойстве **Database** компонента **IBQueryl** устанавливаем **DM.IBDatabasel**. Теперь заходим в свойство **SQL** этого же компонента, появится окно **Command Text Editor**. В этом окне пишем запрос, как показано на рисунке. Данный запрос означает выбрать все данные из таблицы «**Товар**» и отсортировать их по названию товара.



18. Щелкаем два раза левой кнопкой мыши по компоненту **IBQuery1**. Щелкаем правой кнопкой мыши в появившемся окне и выбираем **Add all fields.** После этого действия окно примет вид:

19. Выбираем **TOVARID** и в инспекторе объектов свойство **Visible** ставим в **False**. Теперь этот атрибут не будет отображаться на экране в сетке **DBGrid** во время работы с программой.
20. Для атрибута **Name** в свойство **DispiayLabel** пишем «**Название товара**». В свойства **DipiayWidth** ставим 30 — это свойство отвечает за ширину атрибута в **DBGrid**.
21. Выбираем атрибут **Price**, в свойство **DispiayLabel** пишем «**Цена**».
22. Щелкаем два раза на кнопке **Button** с текстом «**Закрыть**» и пишем код для закрытия формы «**Справочник — товары**»: **FormTovar. Close;**
23. Выбираем пункт меню **Project - Option** и вкладку **Forms**. И с помощью стрелок переносим **FormTovar** в правую половину окна. Таким образом, мы говорим **Delphi**, что во время работы программы мы будет сами следить за созданием этой формы и ее удалением из памяти компьютера. Нажимаем **ОК**.



24. Делаем активной форму **MainForm** и для пункта меню **Справочники - Товар** пишем код, предварительно подключив модуль **UTovar** (это делается с помощью пункта меню **File - Use Unit**):

FormTovar:=TFormTovar.Create(self); {Создаем форму}

FormTovar.IBQueryl.Open; {Открываем набор данных}

FormTovar.ShowModal; {Отображаем форму на экране}

25. Переходим на форму **FormTovar** и в событии **OnClose** пишем код для закрытия набора данных перед закрытием формы: **FormTovar.IBQueryl.Close**;
26. Помещаем на форму **FormTovar** еще один компонент **IBQuery** (он будет называться **IBQuery2**), в свойство **Database** пишем **DM.IBDatabase1**.
27. В свойстве **SQL** компонента **IBQuery2** пишем запрос для добавления данных в таблицу «**Товар**»: **INSERT INTO TOVAR (NAME, PRICE) VALUES (:NAME, :PRICE)**. Данный запрос означает добавить новую запись в таблицу **Tovar**, а именно в атрибуты **Name** и **Price**, которые содержатся в параметрах **:Name** и **:Price**. Параметры легко можно отличить от остальных элементов по двоеточию стоящему перед их именем.
28. Щелкаем по свойству **Params** компонента **IBQuery2**, появляется окно **Editing.**

29. Выделяем запись «**0 – Name**» и в инспекторе объектов ставим в свойстве **DataType** значение **ftString**. В свойстве **Value** указываем «**AAA**». **DataType** — тип данных, который будет содержать параметр, в данном случае текстовый, **Value** — значение по умолчанию, если параметру не присвоено никакого значения (предположим, произошел какой-то сбой в системе) ему будет присвоено «**AAA**».
30. Выделяем запись «**1 – Price**» и в инспекторе объектов ставим в свойство **DataType** значение **ftInteger** — значение параметра целое число, а в свойство **Value** ставим значение по умолчанию **0**.
31. Располагаем на форме «**Товар**» еще одну кнопку **Button**, в свойстве **Caption** пишем «+». В событие **onClick** пишем код для добавления новой записи в таблицу «**Товар**»:

```
procedure TFormTovar.Button3Click(Sender: TObject);
var
s:string; {для названия товара}
c:integer; {для цены товара}
begin
{Используем стандартную функцию InputBox, которая выводит на экран окно с полем для ввода информации}
s:=InputBox('Введите данные','Введите название товара',' ') ;
c:=StrToInt(InputBox('Введите данные','Введите цену товара',' '));
{Передаем в параметр Name введенное название товара}
IBQuery2.Params.ParamByName('Name').Value:=s;
{Передаем в параметр Price введенную цену товара}
IBQuery2.Params.ParamByName('Price').Value:=c;
try
{Выполняем запрос, методом ExecSQL, т.к. запрос не будет возвращать данных}
IBQuery2.ExecSQL;
```

{Если не удалось, то сообщить пользователю}

except

{Выводим окно с текстом с помощью функции ShowMessage}

ShowMessage('Ошибка при добавлении данных!'+#13+ 'Попробуйте еще раз!');

{откатываем изменение}

DM.IBTransaction1.RollbackRetaining;

{выходим из процедуры с помощью exit}

exit;

{Если запрос выполнился, то подтверждаем транзакцию}

DM.IBTransaction1.CommitRetaining;

{Обновляем набор данных}

IBQuery1.Close;

IBQuery1.Open;

end;

end;

32. Помещаем на форму еще один компонент **IBQuery** (его имя будет **IBQuery3**), свойство **DataBase** устанавливаем в **DM.IBDatabase1**.
33. В свойстве **SQL IBQuery3** пишем код для удаления записи из таблицы «**Товар**»:

DELETE FROM Tovar WHERE TovarID=:Par1;

Данный запрос означает: удалить из таблицы **Tovar** товар с кодом (запись, у которой атрибут **TovarID**), равным значению параметра.

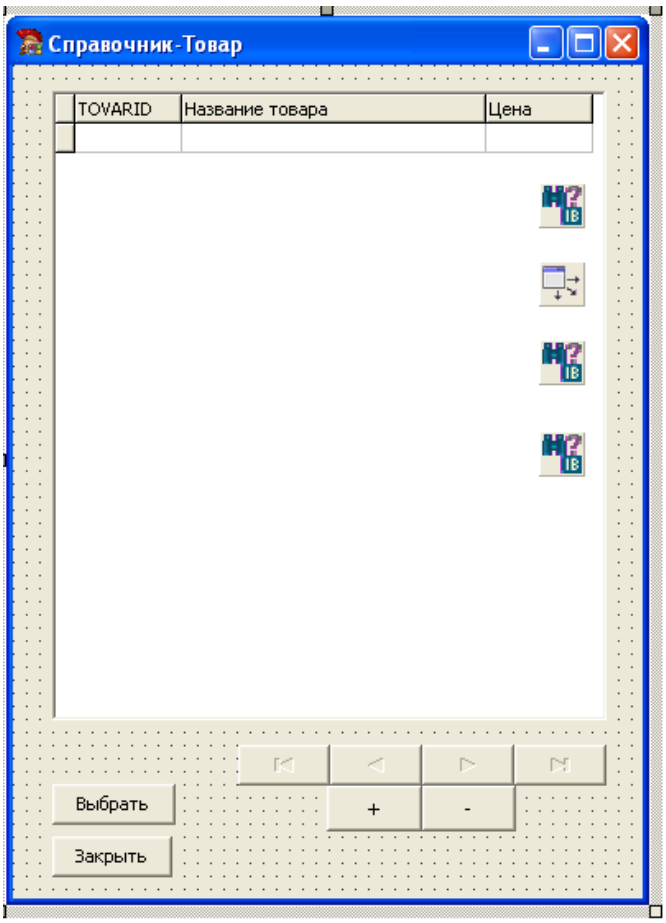
34. Щелкаем по свойству **Params** компонента **IBQuery3**, появляется окно **Editing.**

35. Выделяем запись «**0 –Par1**» и в инспекторе объектов ставим в свойстве **DataType** значение **ftInteger**, в свойстве **Value** пишем значение по умолчанию 0.
36. Помещаем на форму еще одну кнопку **Button**, в свойство **Caption** пишем «-». В событие **OnClick** пишем код удаления записи из таблицы «**Товар**»:

```
procedure TFormTovar.Button4Click(Sender: TObject);
begin
 {Спрашиваем пользователя,  правда ли он хочет удалить запись,
используется стандартная функция MessageDlg для вывода
окна  с кнопками Yes и No}
 if MessageDlg('Вы уверены, что хотите удалить запись?',
mtConfirmation,[mbYes,mbNo],0)=mrYes
 then
{если да,   то}
begin
{запоминаем код выбранной записи и помещаем  это значение в параметр Parl}
IBQuery3.Params.ParamByName('Par1').Value:= IBQuery1TovarID.Value;
{Пытаемся удалить запись}
try
{Выполняем запрос методом ExecSQL,т.к. запрос не будет возвращать данных}
IBQuery3.ExecSQL;
{если не получилось, то сообщаем об этом пользователю}
except
{Выводим окно с текстом с помощью функции ShowMessage}
ShowMessage ( 'Удаление не прошло !' +#13+ 'Запись заблокирована либо уже
удалена!');
{открываем транзакцию}
DM.IBTransaction1.RollbackRetaining;
{выходим из процедуры с помощью exit}
exit;
end;
{Если все нормально, то подтверждаем транзакцию}
DM.IBTransaction1.CommitRetaining;
{Обновляем набор данных}
IBQuery1.Close;
IBQuery1.Open;
end;
```

end;

37. Полученная форма приведена на рисунке:



38. Создаем новую форму, называем ее **FormFirm**. В свойство **Caption** пишем **«Справочник – фирмы»**. Сохраняем под именем **UFirm**.
39. Помещаем компоненты **DBGrid**, **DBNavigator**, 2 компонента **Button**, **IBQuery, DataSource**.
40. Свойство **Enabled** кнопки «**Выбрать**» установите в **False**. Для компонента **DBNavigator** свойство **VisibieButtons** сделайте таким же, как в прошлый раз. В свойство **DataSet** компонента **DataSource** устанавливаем **IBQuery1**.
41. В свойство **DataSource** компонентов **DBGrid** и **DBNavigator** ставим **DataSource1**.
42. Выбираем пункт меню **File - USE Unit**. В появившемся окне **Use Unite** выбираем **UDM** и нажимаем **ОК**. Это нужно, чтобы мы смогли подключить компонент **IBQuery1** к компоненту **IBDatabase**, так как они находятся на разных формах. В свойстве **Database** компонента **IBQuery1** ставим **DM.IBDatabase1**.

43. Теперь заходим в свойство **SQL** этого же компонента, и пишем в окне **Command Text Editor** запрос для выборки всех записей из таблицы «**Фирмы**»:

SELECT * FROM Firm ORDER BY Name

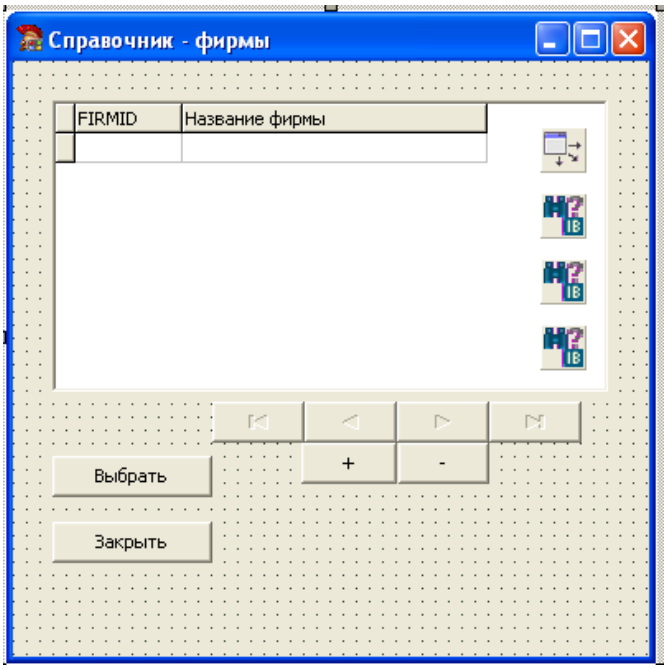
Данный запрос означает выбрать все данные из таблицы «**Фирмы**» и отсортировать их по названию фирмы (атрибут **Name**).

44. Щелкаем два раза левой кнопкой мыши по компоненту **IBQueryl** и добавляем поля к данному компоненту. После этого приступим к настройке отображения атрибутов.
45. Выбираем **FirmID** и в инспекторе объектов в свойство **Visible** ставим **False**. Теперь этот атрибут не будет отображаться на экране во время работы программы.
46. Для атрибута **Name** в свойство **DisplayLabel** пишем «**Название фирмы**». В свойство **DispiayWidth** ставим 30 — это свойство отвечает за ширину поля в **DBGrid**.
47. Щелкаем два раза на кнопке **Button** с текстом «**Закрыть**» и пишем код для закрытия формы «**Справочник — фирмы**»: **FormFirm. Close;**
48. Выбираем пункт **Project - Option** главного меню. И с помощью стрелок переносим **FormFirm** в правую половину окна; таким образом, мы указываем **Delph**i, что во время работы программы мы будем сами следить за созданием этой формы и ее удалением. После этого нажимаем **ОК**.
49. Делаем активной форму **MainForm** и для пункта меню **Справочники - Фирмы** пишем код для создания формы «**Справочник — фирмы**» (предварительно надо будет подключить модуль **UFirm**):

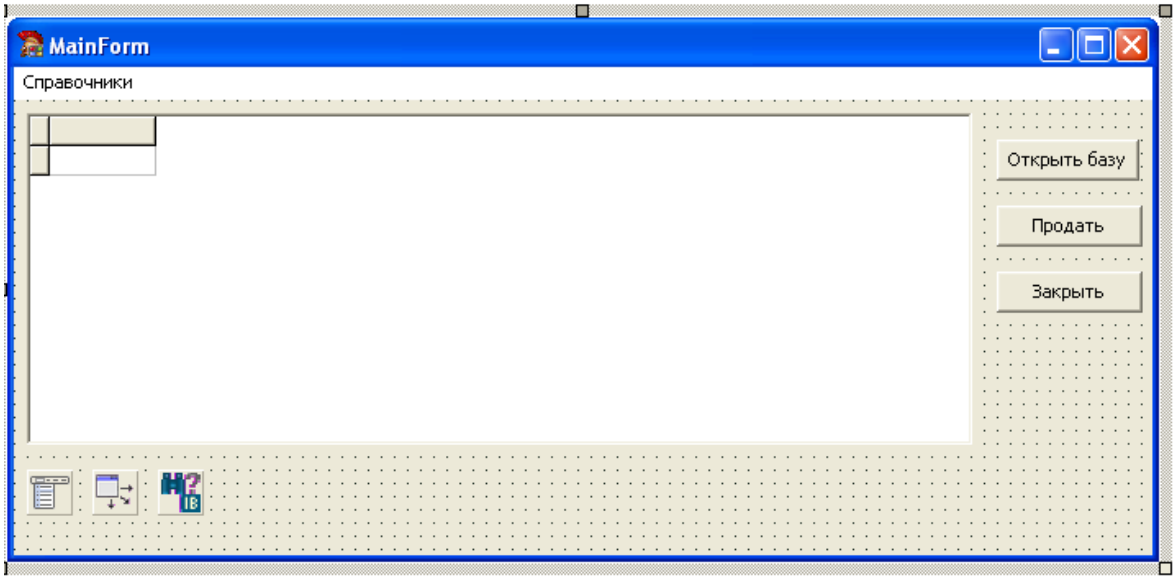
```
FormFirm:=TFormFirm.Create(self); {Создаем форму}
FormFirm.IBQueryl.Open; {Открываем набор данных}
FormFirm.ShowModal; {Отображаем форму на экране}
```

50. Переключаемся на форму **FormFirm** и в событии **OnClose** пишем: **FormFirm.IBQueryl.Close**;
51. Помещаем на форму компонент **IBQuery** (он будет называться **IBQuery2**), в свойство **DataBase** пишем **DM.IBDatabase1**. В свойство **SQL** пишем запрос для добавления данных в таблицу «**Фирма**»: **INSERT INTO Firm (Name) VALUES (:Name);**
52. Щелкаем по свойству **Params** компонента **IBQuery2**, в появившемся окне **Editing** выделяем запись **Name** и в инспекторе объектов ставим в свойство **DataType** значение **ftString**, в свойство **Value** ставим значение по умолчанию «**ББВ**».

53. Располагаем на форме еще одну кнопку **Button3.** В свойство **Caption** пишем «+». В событие **OnClick** кнопки пишем код для новой записи в таблицу «**Фирмы**».
54. Помещаем на форму еще один компонент **IBQuery** (его имя будет **IBQuery3**), в свойство **DataBase** устанавливаем **DM. IBDatabase1**.
55. В свойстве **SQL** компонента **IBQuery3** пишем запрос для удаления данных из таблицы «**Фирмы**»: **DELETE FROM Firm WHERE FirmID=:Par1**
56. Щелкаем по свойству **Params** компонента **IBQuery3**, появляется окно **Editing**. Выделяем запись «**0 - Par1**» и в инспекторе объектов ставим в свойстве **DataType** значение **ftInteger**, в свойстве **Value** пишем значение по умолчанию **0**.
57. Помещаем на форму еще одну кнопку **Button**, в свойство **Caption** пишем «**-**», В событие **onClick** пишем код для удаления записи из таблицы «**Фирмы**».
58. На рисунке представлен окончательный вариант формы «**Справочник – фирмы**».



59. Переходим на форму **MainFom**, располагаем на ней компоненты **DBGrid**, **IBQuery**, **Datasource** и три компонента **Button.**
60. У кнопок **Button** в свойстве **Caption** пишем: «**Открыть базу», «Продать», «Закрыть**», как показано на рисунке.



61. Выбираем пункт меню **File - USE Unit**. Выбираем **UDM** и нажимаем **ОК**.
62. В свойстве **Database** компонента **IBQuery**l устанавливаем **DM.IBDatabasel**.
63. Теперь заходим в свойство **SQL** этого же компонента и напишем запрос для выборки записей из таблиц «**Продажи», «Товар», «Фирмы**» по условию:

```
SELECT  *   FROM Sale,Tovar,Firm
WHERE
Sale.FIRMKOD=FIRM.FIRMID
AND
Sale.TOVARKOD=TOVAR.TOVARID
```

Данный запрос означает выбрать все данные по продажам. Причем нужно связать таблицы **Sale** и **Firm** по коду организации, а таблицы **Sale** и **Tovar** - связать по коду товара. Таким образом, с помощью этого запроса мы связываем все три таблицы.

64. Щелкаем два раза по компоненту **IBQueryl** и настраиваем атрибуты для отображения в приложении.
65. В свойство **DataSet** компонента **DataSource** устанавливаем значение **IBQuery1**. А в свойство **DataSource** компонента **DBGrid** устанавливаем **DataSourcel**.
66. Щелкаем два раза по кнопке **Button** с текстом «**Открыть базу**» и здесь пишем простую процедуру для аутентификации пользователя при попытке открыть таблицу «**Продажи**»:

```
procedure TMainForm.Button1Click(Sender: TObject);
var
```

```
s:string; {Хранит введенный пароль}
begin
{Используем стандартную функцию InputBox,
которая выводит на экран окно с полем для
ввода информации; таким образом мы
будем запрашивать пароль для открытия базы}
s:=InputBox('Аутентификация','Введите пароль:','');
{если пароль правильный (паролем является слово Parol), то}
if s='Parol' then
begin
{открываем базу продаж}
IBQuery1.Open;
{кнопку "Открыть базу" делаем недоступной}
Button1.Enabled:=false;
end
else
{если пароль неправильный, то выводим
пользователю информационное сообщение}
ShowMessage('Неправильный пароль!');
end;
```

Теперь случайный человек не сможет получить доступ к базе продаж.

67. В событие **OnClose** формы **MainForm** пишем код для закрытия базы продаж при выходе из программы:

```
procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
{Если выходим из программы и база продаж открыта,
то закрыть ее}
if IBQuery1.Active then IBQuery1.Close;
end;
```

68. Располагаем на форме **MainForm** компонент **IBQuery**, задаем ему свойство **DatabaseName**, в свойстве **SQL** пишем запрос для добавления данных в таблицу «Продажи»:

INSERT INTO SALE (FIRMKOD, TOVARKOD, REM)

VALUES (: FIRMKOD, : TOVARKOD, :REM) ;

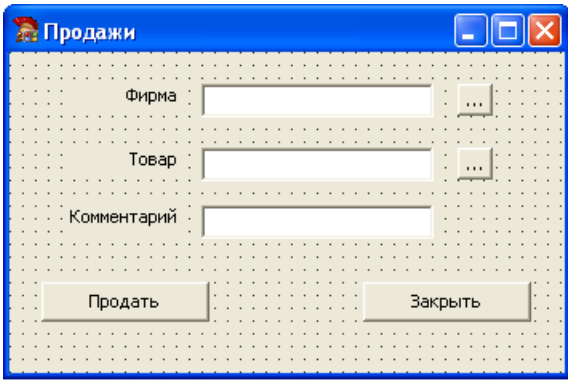
69. Задаем свойство **Params**. Первые два параметра типа **ftInteger** со значением по умолчанию 0. Третий параметр **ftString**, значение по умолчанию — «**BBB**».
70. Для кнопки **Button** с текстом «**Продать**» пишем код для создания формы **FormSale**:

{Создание и отображение формы FormSale}

FormSale:=TFormSale.Create(self);

FormSale.ShowModal;

71. Создаем новую форму, называем ее **FormSale**, сохраняем под именем **USale**. Располагаем на ней компоненты, как показано на рисунке (компонент с тремя точками — это обычный **Button**).



72. Выбираем пункт главного меню **Project - Option**, отменяем автоматическое создание формы.
73. Для события **OnClick** кнопки **Button** с тремя точками, расположенной напротив слова «**Фирма**», пишем код для создания формы **FormFirm:**

FormFirm:=TFormFirm.Create(self);

FormFirm.IBQueryl.Open;

{Делаем доступной кнопку "Выбрать"}

FormFirm.Buttonl.Enabled:=True;

FormFirm.ShowModal;

74. Выбираем форму **FormFirm** и для кнопки **Button** с текстом «**Выбрать**» пишем код:

procedure TFormFirm.ButtonlClick(Sender: TObject);

begin

{Запоминаем название организации и его код, код

пишем в свойство Tag}

```
FormSale.Edit1.Text:=IBQuery1NAME.Value;
FormSale.Edit1.Tag:=IBQuery1FIRMID.Value;
FormFirm.Close;
end;
```

75. Выбираем форму **FirmSale**. Для кнопки с тремя точками, расположенной напротив слова «**Товар**», пишем код для создания формы **FormTovar**:

```
FormTovar:=TFormTovar.Create(self);
FormTovar.IBQuery1.Open;
{Делаем доступной кнопку "Выбрать"}
FormTovar.Button1.Enabled:=True;
FormTovar.ShowModal;
```

76. Выбираем форму **FormTovar** и для кнопки «**Выбрать**» пишем код:

```
procedure TFormTovar.Button2Click(Sender: TObject);
begin
{Запоминаем название товара и его код, код
пишем в свойство' Tag'}
FormSale.Edit2.Text:=IBQuery1NAME.Value;
FormSale.Edit2.Tag:=IBQuery1TOVARID.Value;
FormTovar.Close;
end;
```

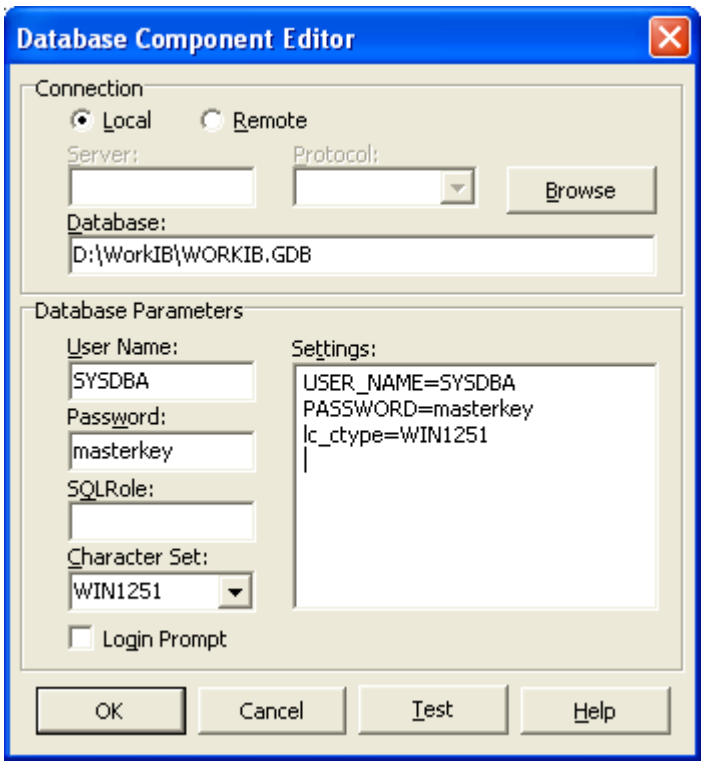
77. Переходим на форму **FormSale**, для кнопки «**Закрыть**» пишем код: **FormSale.Close**;

78. Для кнопки «**Продать**» пишем код для продажи товара:

```
procedure TFormSale.Button3Click(Sender: TObject);
begin
{Передаем в параметры данные по продаже}
MainForm.IBQuery2.Params.ParamByName('FIRMKOD').Value:=Edit1.Tag;
MainForm.IBQuery2.Params.ParamByName('TOVARKOD').Value:=Edit2.Tag;
MainForm.IBQuery2.Params.ParamByName('REM1').Value:=Edit3.Text;
{пытаемся выполнить запрос}
try
MainForm.IBQuery2.ExecSQL;
{если не удалось, то сообщаем об сшибке}
```

```
except
ShowMessage('Продать товар не цолучилось!'+#13+ 'Повторите попытку!');
{откатываем транзакцию}
DM.IBTransaction1.RollbackRetaining;
exit;
end;
{если запрос выполнился, то подтверждаем транзакцию}
DM.IBTransaction1.CommitRetaining;
{обновляем таблицу продаж, если она открыта}
if MainForm.IBQuery1.Active then
begin
MainForm.IBQuery1.Close;
MainForm.IBQuery1.Open;
end;
FormSale.Close;End;
```

79. Пишем обработчик для кнопки «**Закрыть**», расположенной на форме **MainForm**: **MainForm.Close;**
80. Теперь щелкнем два раза на компоненте **IBQuery1**, расположенном на главной форме, добавляем все атрибуты и оставляем видимыми только те, что нужны нам: **Rem, Name, Name1, Price**. Можно менять их очередность, перетаскивая мышкой. Задаем им русский текст в свойстве **DispiayLabe**.
81. Перейдите на **UDM (DataModule)**, щелкните правой кнопкой мыши по компоненту **IBDatabase** и выберите пункт **Database Editor**, появится окно **Database Component Editor**.
82. В выпадающем списке **Character Set** обязательно выберите **WIN1251**, без этого не будут нормально отображаться русские символы.



83. Для того чтобы подключиться с другого компьютера к нашей базе данных, надо установить на этот компьютер клиента **InterBase** и написать в программе процедуру **openBD**.
84. Для этого сделайте активной форму **MainForm**, нажмите **F12** и перед разделом **var** объявите процедуру **OpenBD**:

```
{Процедура открытия базы}
procedure OpenBD;
```

85. Для открытия базы с использованием пути из файла пишем код:

```
procedure OpenBD;
Var
  FP:TextFile;
  StringPath:string;
  NameOfFile:string;
begin
{получаем путь к файлу path.txt,  там у нас будет
Храниться адрес БД}
NameOfFile:=ExtractFileDir(Application.ExeName)+'\path.txt';
{если файл есть,  то связываемся с ним, иначе
```

```
Сообщаем об ошибке и выходим из программы}
if FileExists (NameOfFile)   then
begin;
Assign (Fp, NameOfFile);
try
begin
Reset(FP);
Read(FP,StringPath);
End;
finally
CloseFile(FP);
End
End
else
begin
ShowMessage('Файл пути к БД не найден' +#13+ 'Создайте в папке программы файл path.txt' + #13+
 'и запишите туда путь к БД' +#13+   'Пример:   C: \WorkIB\WorkIB. GDB') ;
 Application.Terminated;
end;
{задаем базу данных для  компонента  IBDatabase}
DM.IBDatabase1:=StringPath;
{Открываем базу данных и компонент для работы с
транзакциями БД}
with DM do
begin
IBDatabase1.Connected:=True;
IBTransaction1.Active:=True;
end;
end;
```

86. Теперь для модуля данных **DM** в событии **onCreate** вызываем процедуру **openBD** (только необходимо будет подключить модуль **UMain**):

procedure TDM.DataModuleCreate(Sender: TObj ect);

begin

OpenBD;

end;

87. В нашей рабочей папке создадим текстовый файл **path.txt**, можно с помощью **Блокнота**. В нем прописываем путь к базе, например, для случая, когда база лежит локально: **D:\WorkIB\WorkIB.gdb**
88. Если необходимо подключаться к удаленной базе, то в файле **path.txt** необходимо сделать запись вида: **Имя_сервера: имя_файла_бд** (имя сервера может быть символьным, либо можно ввести его IP-адрес).
89. Еще один важный момент. Когда мы выполняем транзакции, мы никогда не пишем **IBTransaction1.StartTransaction**, мы либо подтверждаем их, либо откатываем. Это не ошибка, просто транзакции у нас стартуют автоматически, за это и отвечает компонент **IBTransaction**.

Внеаудиторная самостоятельная работа:

Составить опорный конспект письменно в тетради по основным командам.

Практическая работа №19

Формирование SQL запросов для выборки данных. Простые и сложные запросы на выборку (сортировка, группировка, вычисляемые поля, составные операторы выборки). Создание SQL запросов для изменения наборов данных.

Цель работы: сформировать умения по созданию простых и сложных SQL запросов на выборку данных.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**.

Задание. Построение SQL запросов выборки данных

Методические указания по выполнению задания:

1. Для выборки данных из таблиц используется SQL-оператор **SELECT**, который в простейшей форме имеет следующий синтаксис:

   SELECT столбцы FROM имя_таблицы

2. Для выборки данных из столбцов **Zip** и **City** таблицы **REGIONS** применяют следующий оператор:

   SELECT "Zip", "City" FROM REGIONS

3. Если из таблицы необходимо выбрать все данные, то вместо перечня столбцов можно указать символ *****. Например, **SELECT * FROM REGIONS**

zip	area	region	city
2001	<null>	<null>	Киев
2002	<null>	<null>	Киев
3150	<null>	<null>	Киев
8320	Киевская обл.	<null>	Борисполь
8324	Киевская обл.	Бориспольский р-н	Гора
8342	Киевская обл.	Бориспольский р-н	Ревное

4. В операторе **SELECT** можно определять условия выборки, используя с этой целью ключевое слово **WHERE**. Необходимо выбрать из таблицы **STAFF** данные обо всех руководителях старшего звена, которых зовут **Иванами**:

   SELECT * FROM STAFF WHERE ("DepID" = 1) AND ("FirstName" = 'Иван')

ID	LastName	FirstName	FatherName	Zip	Street	House	Tel
1	Иванов	Иван	Иванович	2001	<null>	<null>	<null>
2	Петров	Иван	Васильевич	2002	<null>	<null>	<null>

5. Ключевое слово **LIKE** позволяет выполнить поиск данных по шаблону. В качестве

шаблона поиска стандартом SQL определены два подстановочных знака, означающих следующее: **%** — любое количество символов в текущей позиции; **_** - одиночный символ в текущей позиции.

6. Ниже следующий оператор SQL вернет четыре строки, соответствующие трем Ивановым и одной Ивановой.

   SELECT * FROM STAFF WHERE "LastName" LIKE 'Иванов%'

7. В то же время ниже представленный запрос вернет только строку, соответствующую фамилии Иванова.

   SELECT * FROM STAFF WHERE "LastName" LIKE 'Иванов_'

8. Предположим, нам необходимо выяснить, в каких населенных пунктах проживают сотрудники организации. Выполним следующий SQL-оператор:

   SELECT "City" FROM REGIONS

   В результате будут возвращены шесть строк.

city
Киев
Киев
Киев
Борисполь
Гора
Ревное

   Однако нам требуется просмотреть только неповторяющиеся значения из столбца **City**, чтобы одно и то же название населенного пункта не дублировалось в результате запроса несколько раз. Для этого модифицируем предыдущий оператор **SELECT**:

   SELECT DISTINCT "City" FROM REGIONS

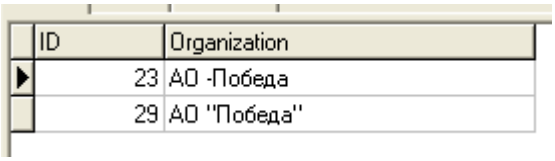
   Ключевое слово **DISTINCT** используется для выбора строк, в которых значения в указанных полях не повторяются.

city
Борисполь
Гора
Киев
Ревное

9. Обратите внимание на то, что в случае использования ключевого слова **DISTINCT** выбранные данные автоматически сортируются в порядке возрастания. После слова **DISTINCT** можно указать и несколько столбцов. Так, если мы хотим просмотреть сведения о трудовой деятельности сотрудников, то должны выполнить следующий SQL-

запрос:

SELECT DISTINCT "ID", "Organization" FROM JOBS

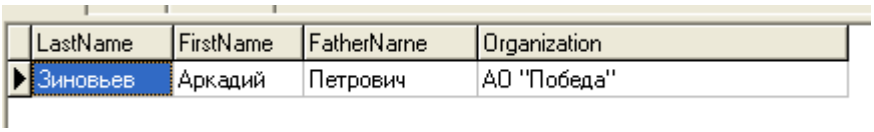


ID	Organization
23	АО -Победа
29	АО "Победа"

10. Данные, отобранные предыдущим запросом неудобны в работе, поскольку вместо имени сотрудника отображается его уникальный код. Чтобы исправить эту ситуацию, необходимо извлечь соответствующие данные из столбцов **LastName, FirstName** и **FatherName** таблицы **STAFF**. Для этого используется еще одна функция ключевого слова **WHERE**, называемая объединением таблиц. Операция объединения выполняется в два этапа: с помощью ключевого слова **FROM** в операторе **SELECT** перечисляются объединяемые таблицы, объединение этих таблиц по необходимому полю (или полям) с помощью ключевого слова **WHERE**.
11. В рассмотренном выше примере это должно выглядеть так:

    SELECT DISTINCT STAFF."LastName", STAFF."FirstName", STAFF."FatherName", JOBS."Organization" FROM STAFF, JOBS WHERE STAFF."ID" = JOBS."ID"

    В результате выполнения этого оператора будут возвращены следующие строки.



LastName	FirstName	FatherName	Organization
Зиновьев	Аркадий	Петрович	АО "Победа"

    Таблица, указанная в условии **WHERE** слева от знака равенства, называется внешней (outer), а таблица, указанная справа, — внутренней (inner).
12. Объединения, подобные рассмотренному выше, называются **внутренними** (**INNER JOIN**), или правыми (**RIGHT JOIN**). Такие объединения возвращают только те строки, для которых выполняется условие **WHERE**. Для создания правых объединений возможно также использование записи вида:

    SELECT Таблица 1.Столбец1, Таблица2. Столбец2

    FROM Таблица1 INNER JOIN Таблица2

    ON Таблица1.Столбец1 = Таблица2.Столбец2
13. Обычно для наглядности вместо ключевого слова **INNER** применяют ключевое слово **RIGHT**. Например, представленный на рисунке набор строк можно было бы получить с помощью следующего SQL - запроса:

SELECT DISTINCT STAFF."LastName", STAFF."FirstName",

STAFF."FatherName", JOBS."Organization"

FROM STAFF RIGHT JOIN JOBS

ON STAFF."ID" = JOBS."ID"

14. Если в таблицах, которые участвуют в объединении, названия столбцов не пересекаются (как в случае с таблицами **STAFF** и **JOBS**), то SQL-оператор можно упростить, опустив префиксы после ключевых слов **SELECT** и **WHERE**.
15. Объединения могут быть также **внешними** (**OUTER JOIN**), или левыми (**LEFT JOIN**), т.е. такими, которые возвращают строки вне зависимости от выполнения условия объединения. При использовании внешних объединений все поля внутренней таблицы, не соответствующие условию объединения, возвращаются со значением **NULL**. Синтаксис для создания внешних объединений имеет следующий вид:

    SELECT Таблица1.Столбец1, Таблица2.Столбец2

    FROM Таблица1 LEFT JOIN Таблица2

    ON Таблица1.Столбец1 = Таблица2.Столбец2

    Ключевое слово **OUTER**, хотя и определено стандартом SQL, многими системами управления базами данных не поддерживается. Предпочтение отдается использованию слова **LEFT**.
16. Например, в рассмотренном объединение таблиц **STAFF** и **JOBS** заменим ключевое слово **RIGHT** на **LEFT**:

    SELECT DISTINCT STAFF."LastName", STAFF."FirstName",

    STAFF."FatherName", JOBS."Organization"

    FROM STAFF LEFT JOIN JOBS

    ON STAFF."ID" = JOBS."ID"

    Полученный результат показан на рисунке.

LastName	FirstName	FatherName	Organization
Зиновьев	Аркадий	Петрович	АО "Победа"
Иванов	Иван	Иванович	<null>
Петров	Иван	Васильевич	<null>

    Как видим, в строках, для которых отсутствует соответствие таблиц **STAFF** и **JOBS**, в поле **Organization** находится значение **NULL**.
17. Еще один способ объединения данных, извлеченных из нескольких таблиц, — многотабличные запросы, возвращающие декартово произведение всех столбцов (т.е все

комбинации значений из одной таблицы и значений из другой таблицы). Такие объединения называются **полными.** Например, выполним следующий запрос:

SELECT * FROM REGIONS, FAMILY

В результате выполнения запроса мы получили объединение первой строки из таблицы **REGIONS** со всеми строками таблицы **FAMILY**, затем — объединение второй строки из таблицы **REGIONS** со всеми строками таблицы **FAMILY** и т.д.

18. В таких запросах, как и в любых других, можно выбирать только отдельные поля таблиц, использовать условие **WHERE** и внутренние/внешние объединении. Например:

    SELECT REGIONS."Zip", REGIONS."City", STAFF."LastName", STAFF."FirstName", STAFF."FatherName", FAMILY."Kin", FAMILY."KinName"
    FROM REGIONS, STAFF, FAMILY
    WHERE (REGIONS."City" <> 'Киев') AND
    (FAMILY."Kin" IN ('Сын', 'Дочь')) AND
    (STAFF."ID" = FAMILY."ID")

19. **Вложенные запросы** — это операторы **SELECT**, используемые в той части запроса, которая определяется ключевым слоном **WHERE**. Такие запросы применяются для предварительной выборки данных, используемой затем основным оператором **SELECT.** Для этого применяется конструкция вида:

    SELECT столбцы FROM таблицы
    WHERE (условия_выборки/объединения) AND
    (столбец IN (SELECT ...))

20. Например, выберем имена и названия должностей только таких сотрудников, которые занимают руководящие посты:

    SELECT STAFF."LastName", STAFF."FirstName",
    STAFF."FatherName", POSS."PosFullName"
    FROM STAFF, POSS
    WHERE (STAFF."PosID" = POSS."PosID") AND
    (STAFF."PosID" IN (SELECT "PosID" FROM POSS
    WHERE "PosLevel" BETWEEN 1 AND 3))

    В этом операторе сначала выполняется вложенный запрос, в котором выбираются идентификаторы должностей уровня 1. 2 или 3. Затем выбираются данные из таблицы **STAFF**, у которых значение поля **PosID** входит в набор ранее выбранных

идентификаторов должностей. Дополнительно выполняется объединение таблиц **STAFF** и **POSS** по столбцу **PosID**, чтобы в результирующем наборе данных вместо числового идентификатора отображались фактические названия должностей.

Полученный результат представлен ниже.

LastName	FirstName	FatherName	PosFullName
Петров	Иван	Васильевич	Заместитель ген. директора
Иванов	Иван	Иванович	Генеральный директор

21. В представленных выше примерах имена столбцов в результирующих наборах данных соответствуют фактическим именам столбцов в таблицах. Однако это не всегда удобно. Язык SQL позволяет использовать имена логических псевдостолбцов, называемые **псевдонимами столбцов**. Их назначение — сделать текст запроса более понятным, а полученный набор данных — нагляднее. Псевдонимы столбцов размещаются в операторе **SELECT** справа от имен полей после ключевого слова **AS**. При этом допускается использование пробелов и употребление символов кириллицы. В том случае, если в псевдониме используются пробелы, он должен быть заключен в кавычки (в InterBase — только в двойные). Если же в псевдониме применяются символы кириллицы (без пробелов), то в InterBase они должны быть взяты в двойные кавычки.
22. В списке, определяемом ключевым словом **WHERE**, псевдонимы не допускаются. Здесь всегда необходимо указывать реальные имена столбцов.
23. Внесем коррективы в последний SQL-запрос:

```
SELECT STAFF."LastName" AS "Фамилия",
       STAFF."FirstName" AS "Имя",
       STAFF."FatherName" AS "Отчество",
       POSS."PosFullName" AS "Полное название организации"
FROM STAFF, POSS
WHERE (STAFF."PosID" = POSS."PosID") AND
(STAFF."PosID" IN (SELECT "PosID" FROM POSS
WHERE "PosLevel" BETWEEN 1 AND 3))
```

При этом двойные кавычки в заголовках столбцов не отображаются.

Фамиия	Имя	Отчество	Полное название организации
Петров	Иван	Васильевич	Заместитель ген. директора
Иванов	Иван	Иванович	Генеральный директор

24. В качестве источника данных в операторе **SELECT** можно указывать не только имена столбцов таблицы. Допустимы также выражения, содержащие математические операции, абсолютные значения и вызовы функций.
25. Выберем, например, имена только таких сотрудников, у которых есть несовершеннолетние дети (моложе 18 лет). Для этого воспользуемся средой InterBase, поскольку в ней поддерживается стандартная функция **CURRENT_DATE**, возвращающая текущую дату. Соответствующий запрос будет выглядеть следующим образом:

```
SELECT STAFF."LastName" AS "Фамилия",
    STAFF."FirstName" AS "Имя",
    STAFF. "FatherName" AS "Отчество",
    FAMILY."Kin" AS "Сын/Дочь",
    FAMILY."KinName" AS "Имя ребенка",
    FAMILY."BirthDate" AS "Дата рождения ребенка",
   (CURRENT_DATE - FAMILY."BitrhDate")/365 AS "Возраст ребенка"
FROM STAFF, FAMILY
WHERE (STAFF."ID" = FAMILY."ID") AND (FAMILY."Kin" IN ('Сын','Дочь')) AND
((CURRENT_DATE - FAMILY."BirthDate")/365 < 18)
```

26. **Вычисляемые столбцы** — это выражения, выполняющие те или иные действия над наборами данных. Для этих целей используются функции **COUNT, SUM, AVG, MAX** и **MIN**, которые также называют агрегатными.
27. Функция **COUNT** выполняет подсчет строк. Так, в результате выполнения оператора **SELECT COUNT FROM REGIONS** будет возвращено число шесть - количество строк в таблице **REGIONS**.
28. Подсчитаем количество сотрудников, занимающих руководящие должности:

```
SELECT COUNT(*) FROM STAFF
WHERE "PosID" IN
(SELECT "PosID" FROM POSS WHERE "PosLevel" BETWEEN 1 AND 3)
```

    В результате выполнения этого оператора будет возвращено число 13.
29. Функция **SUM** суммирует значения указанного в ней поля для всех выбранных строк. Подсчитаем суммарную зарплату руководителей старшего звена:

SELECT SUM("Salary") FROM STAFF

WHERE "DepID" = 1

30. Вычислим среднюю зарплату сотрудников по всей организации:

    SELECT SUM("Salary") / COUNT(*) AS

    "Средняя зарплата" FROM STAFF

31. То же самое действие можно выполнить с помощью функции **AVG**, которая определяет среднее арифметическое значений полей всех выбранных строк:

    SELECT AVG("Salary") AS "Средняя зарплата" FROM STAFF.

32. Функция **MAX** определяет наибольшее значение поля среди выбранных строк, а функция **MIN** - наименьшее. Для примера выберем имена и названия должностей сотрудников с наибольшей и наименьшей зарплатой:

    SELECT STAFF."LastName", STAFF."FirstName", STAFF."FatherName",

    POSS."PosFullName", STAFF."Salary"

    FROM STAFF, POSS

    WHERE (STAFF."PosID" = POSS."PosID") AND

    ((STAFF."Salary" IN (SELECT MAX("Salary") FROM STAFF)) OR

    (STAFF."Salary" IN (SELECT MIN("Salary") FROM STAFF)))

    В этом SQL-операторе вначале выполняются вложенные запросы, выбирающие значения наибольшей и наименьшей зарплаты, а затем эти значения используются в условии выборки сведений о сотрудниках.

33. С помощью операции **GROUP BY** в SQL выполняется группирование данных по определенным полям. Эта операция применяется в том случае, если в качестве последнего поля оператора **SELECT** используется агрегатная функция. Рассмотрим пример:

    SELECT DEPS."DeptFullName" AS "Подразделение",

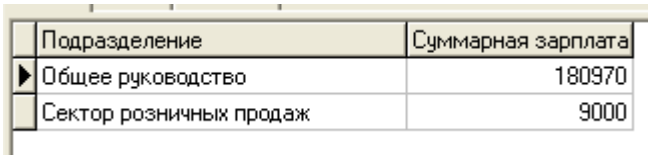
    SUM(STAFF."Salary") AS "Суммарная зарплата"

    FROM DEPS, STAFF

    WHERE STAFF."DepID" = DEPS."DeptID"

    GROUP BY DEPS."DeptFullName"

    В этом операторе выполняется подсчет суммы зарплаты сотрудников с группированием по подразделениям. При этом используется объединение таблиц **STAFF** и **DEPS** по идентификатору подразделения.



Подразделение	Суммарная зарплата
Общее руководство	180970
Сектор розничных продаж	9000

34. Еще один пример использования операции группирования:

    SELECT "LastName" AS "Фамилия",

    COUNT(*) AS "Количество"

    FROM STAFF

    GROUP BY "LastName"

    При помощи такого оператора выполняется подсчет количества однофамильцев в таблице **STAFF**.

35. Ключевое слово **HAVING** используется для выбора строк, возвращаемых в результате выполнения запроса, в котором применяется операция **GROUP BY**. Соотношение ключевых слов **HAVING** и **GROUP BY** примерно такое же, как соотношение ключевого слова **WHERE** и оператора **SELECT**. Отличие состоит только в том, что **HAVING** работает со строками, возвращаемыми запросом, а **WHERE** - со строками исходной таблицы.

    Как правило, запрос, в котором используется слово **HAVING**, можно переписать таким образом, чтобы обойтись без него. Дело в том, что запросы с **HAVING** в большинстве случаев менее эффективны по сравнению с теми запросами, в которых используется только ключевое слово **WHERE.**

36. Пример запроса с использованием ключевого слова **HAVING**.

    SELECT "LastName", COUNT(*) FROM STAFF

    GROUP BY "LastName"

    HAVING "LastName" LIKE 'A%'

    В этом операторе выполняется подсчет однофамильцев, у которых фамилия начинается с буквы «А». Этот же запрос можно записать в другом виде, исключив слово **HAVING.**

37. Операция **ORDER BY** используется для сортировки возвращаемого набора данных. Например:

    SELECT * FROM STAFF

    ORDER BY "LastName", "FirstName", "FatherName"

    В результате выполнения этого запроса будут возвращены все строки таблицы **STAFF**, отсортированные по фамилиям, именам и отчествам.

38. По умолчанию все столбцы, указанные после **ORDER BY**, сортируются в порядке возрастания значений. Порядок сортировки можно изменить для каждого столбца с помощью ключевых слов **ASC** (сортировка по возрастанию) и **DESC** (сортировка по убыванию), например:

    SELECT "LastName", "Salary" FROM STAFF
    WHERE "DepID" = 1
    ORDER BY "LastName" ASC, "Salary" DESC

39. Оператор SELECT можно использовать совместно с оператором INSERT INTO для копирования данных из одной таблицы в другую. В этом случае раздел values заменяется на SELECT, однако правило остается неизменным: поля в списке INSERT INTO должны в точности соответствовать столбцам в наборе данных, полученном с помощью SELECT. К примеру, базу данных **STAFF** можно расширить, добавив в нее отдельную таблицу для хранения сведений об уволенных сотрудниках. Предположим, структура этой таблицы (назовем ее **FIRED**) совпадает со структурой таблицы **STAFF**. В таком случае при увольнении сотрудника соответствующая запись копируется из таблицы **STAFF** в таблицу **FIRED**. Для этого можно воспользоваться следующим оператором:

    INSERT INTO FIRED
    SELECT * FROM STAFF
    WHERE ID = идентификатор_строки

    Затем, конечно же, следует удалить строку из таблицы **STAFF** и записать дату увольнения в поле **StopDate** в соответствующей строке таблицы **JOBS**.

    В том случае, когда таблица **FIRED** включает в себя только часть полей таблицы **STAFF** (например, идентификатор строки, фамилию, имя, отчество, код подразделения и должности, идентификационный и табельный номер, дату рождения и фотографию), то оператор копирования строк примет следующий вид:

    INSERT INTO FIRED
    SELECT "ID", "LastName", "FirstName", "FatherName", "IdCode", "TabNum",
    "BirthDate", "DepID", "PosID"
    FROM STAFF
    WHERE ID = идентификатор_строки

40. Еще один вариант использования подобных операторов копирование строк внутри одной таблицы. Например, в таблице **JOBS** можно копировать данные в столбцах **ID**, **StartDate**

Organization, **Pos** и **Curorg**;

INSERT INTO JOBS("ID", "StartDate", "Organization", "Pos", "CurOrg")

SELECT "ID", "StartDate", "Organization", "Pos", "Curorg"

FROM JOBS

WHERE ("ID" = идентификатор) AND ("StartDate" = дата)

Внеаудиторная самостоятельная работа:

Составить опорный конспект письменно в тетради по основным командам SQL для построения запросов выборки данных.

Практическая работа №20

Создание хранимых процедур. Команды по созданию, редактированию и удалению хранимой процедуры

Цель работы: сформировать умения по выполнению команд создания, редактирования и удаления хранимых процедур в **InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**.

Теоретическая справка:

Хранимая процедура — это программа произвольной длины на языке SQL, которая хранится в клиент-серверной базе данных вместе с другими объектами. Хранимые процедуры позволяют создавать сложные наборы данных, которые не могут быть получены с помощью обычных SQL-операторов. Кроме того, результирующие наборы данных при обращении к удаленным SQL-серверам формируются гораздо быстрее, чем в случае использования обычных операторов SELECT.

Синтаксис, который применяется при создании хранимых процедур, может варьироваться в зависимости от SQL-сервера.

Задание 1. Создание хранимых процедур

Методические указания по выполнению задания:

1. Хранимые процедуры определяют с помощью оператора **CREATE PROCEDURE**, содержащего заголовок и тело процедуры:

   CREATE PROCEDURE заголовок

   BEGIN

   тело__процедуры

   END

2. В структуру заголовка всегда включено имя хранимой процедуры, уникальное внутри базы данных, а в качестве необязательных элементов могут входить перечисленные ниже списки с указанием соответствующих типов данных: входные параметры; выходные параметры; локальные переменные.
3. Тело хранимой процедуры состоит из набора операторов присваивания вида **переменная = выражение**, операторов ветвления, циклических конструкций, операторов **SELECT**, **INSERT, UPDATE**, **DELETE** и специальных команд. Операторы внутри процедуры по умолчанию отделяются друг от друга знаком препинания в виде точки с запятой (;).
4. Хранимые процедуры бывают двух типов: **процедуры выбора** — используются в операторе **SELECT** вместо таблицы и, следовательно, возвращают некоторый набор данных (в заголовке процедуры обязательно должен быть определен набор выходных параметров); выполняемые процедуры — выполняют некоторые операции, однако не обязательно возвращают данные.

Задание 2. Установка разделителя операторов

Методические указания по выполнению задания:

1. По умолчанию разделителем операторов в InterBase является символ «;». При работе с хранимыми процедурами это приводит к конфликтной ситуации, поскольку сама процедура должна распознаваться как отдельный оператор, однако операторы внутри нее также отделяются друг от друга символом «;». По этой причине для выполнения SQL-оператора **CREATE PROCEDURE** необходимо предварительно определить другой символ, который будет использован в качестве внешнего разделителя SQL-операторов. Для этой цели в InterBase предназначен оператор SET TERM.

2. Определим, например, в качестве разделителя SQL-операторов символ «^»:
   SET TERM ^
3. Последовательность SQL-операторов можно записывать в виде сценария. Предположим, мы создаем сценарий, состоящий из двух операторов **SELECT**, двух операторов **CREATE PROCEDURE** и еще двух операторов **SELECT**. Тогда оператор **SET TERM** будет использован следующим образом:
   SELECT ... ;
   SELECT ... ;
   SET TERM ^ ;
   CREATE PROCEDURE ...
   AS BEGIN
   Оператор1;
   оператор2;
   операторN;
   END^
   CREATE PROCEDURE ...
   AS BEGIN
   Оператор1;
   оператор2;
   операторN,
   END^
   SELECT ... ^
   SELECT ...
4. После создания всех процедур можно опять установить прежний разделитель операторов с помощью команды **SET TERM ; ^**.
5. Все блоки операторов внутри хранимой процедуры заключаются между ключевыми словами **BEGIN** и **END**. При этом внутренний разделитель (;) после слова **END** не указывают, а каждый блок операторов может содержать в себе другие блоки.

Задание 3. Входные и выходные параметры

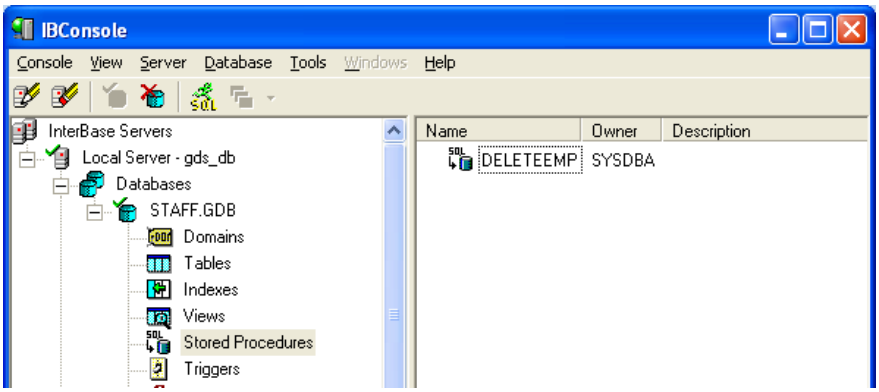
Методические указания по выполнению задания:

1. Входные параметры хранимой процедуры предназначены для передачи в нее данных из

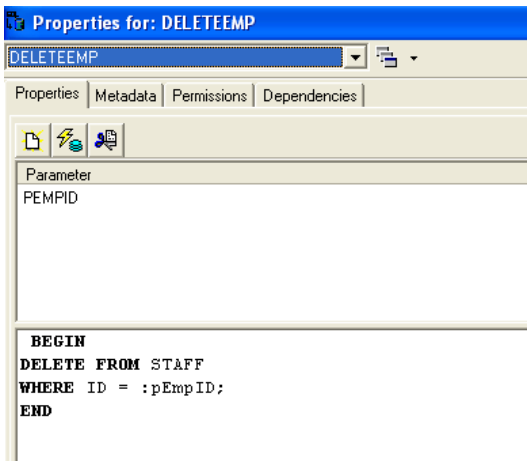
вызывающей программы. Список входных параметров располагают в круглых скобках в заголовке хранимой процедуры после ее имени. При этом названия типов данных отделяются от имени параметра пробелом, а в качестве разделителя параметров используют запятую (,). При обращении к параметру в теле процедуры перед его именем указывают символ двоеточия (:). Например, создадим процедуру **DeleteEmp** для удаления из таблицы **STAFF** строки, в которой поле **ID** содержит значение, передаваемое во входном параметре **pEmpID**:

```
SET TERM ^;
CREATE PROCEDURE DeleteEmp (pEmpID INTEGER)
AS BEGIN
DELETE FROM STAFF
WHERE ID = :pEmpID;
END^
```

2. После выполнения этого SQL-сценария в иерархической структуре базы данных **STAFF.gdb** в категории **Stored Procedures** появится соответствующий элемент.



3. Если на элементе **DELETEEMP** дважды щелкнуть мышью, то откроется диалоговое окно свойств хранимой процедуры. В этом окне на вкладке **Properties** можно просмотреть список параметров и тело процедуры.



Задание 4. Выходные параметры

Методические указания по выполнению задания:

1. Выходные параметры определяют набор данных, возвращаемых хранимой процедурой. Список таких параметров инициализируют в заголовке процедуры с помощью ключевого слова **RETURNS**:

   CREATE PROCEDURE имя_працедуры

   (входные_параметры)

   RETURNS (выходные_параметры)

   AS BEGIN

   END

2. Для определения списка выходных параметров и их использования внутри тела процедуры применяют те же правила, что и для входных параметров.
3. Чтобы записать в выходные параметры результирующие значения, полученные с помощью оператора **SELECT**, используют конструкцию **SELECT … INTO**. Эти значения возвращаются в вызывающую программу в тот момент, когда достигается завершающее слово **END** или в теле хранимой процедуры встречается специальная команда **SUSPEND**.
4. Создадим хранимую процедуру **CountChiefNames**, которая с помощью представления **CHIEFS_1** подсчитывает и возвращает количество фамилий сотрудников на руководящих должностях:

   SET TFRM ^;

   CREATE PROCEDURE CountChieENames

   RETURNS (rNumberOfNames INTEGER)

AS BEGIN

SELECT COUNT(DISTINCT "Фамилия") FROM CHIEFS_1

INTO : rNumberOfNames;

SUSPEND;

END ^

В данном случае входные параметры отсутствуют и определен только один выходной параметр **rNumberOfNames**, которому присваивается результат выполнения оператора **SELECT COUNT(DISTINCT "Фамилия") FROM CHIEFS_1**.

5. Создадим еще одну хранимую процедуру, где применимы одновременно входные и выходные параметры. Эта процедура подсчитывает количество сотрудников и определяет полное имя подразделения, идентификатор которого передается в виде входного параметра. Кроме того, она возвращает полное имя «родительского» подразделения. В представленном ниже фрагменте SQL-кода используются комментарии- поясняющий текст, заключенный между символами **/*** и ***/**. Комментарии можно размещать как в отдельной строке, так и в конце строки с кодом.

SET TERM ^;

CREATE PROCEDURE DepInfo(pDepID INTEGER)

RETURNS (rSize INTEGER, rName VARCHAR(100), rParentName VARCHAR(100))

AS BEGIN

/* Подсчет размера подразделения */

SELECT COUNT(*) FROM STAFF

WHERE “DepID” = :pDepID

INTO :rSize;

/* Определяем название подразделения */

SELECT “DeptFullName” FROM DEPS

WHERE “DeptID “= :pDepID

INTO :rName;

/* Определяем название "родительского" подразделения */

SELECT “DeptFullName” FROM DEPS

WHERE “DeptID” IN (SELECT “ParentDeptID” FROM DEPS

WHERE “DeptID” = :pDepID

INTO :rParentName,-

SUSPEND;

END^

Задание 5. Вызов хранимых процедур

Методические указания по выполнению задания:

1. Для вызова выполняемых хранимых процедур в InterBase используют SQL-оператор **EXECUTE PROCEDURE** вида:

   EXECUTE PROCEDURE имя_процедуры

   список_входных_параметров
2. Для удаления из таблицы STAFF строки, соответствующей Сергею Николаевичу Сушко, можно воспользоваться оператором:

   EXECUTE PROCEDURE DeleCeEmp 21
3. Если внутри выполняемой процедуры были определены выходные параметры, то список соответствующих им параметров в вызывающем SQL-операторе задают после ключевого слова **RETURNING_VALUES**:

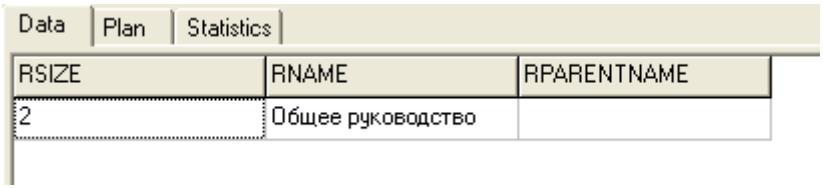
   EXECUTE PROCEDURE имя_процедуры

   список_входных_параметров

   RETURNING_VALUES список_выходных_параметров
4. Для вызова процедур выбора можно использовать или оператор **EXECUTE PROCEDURE**, или же обычный оператор **SELECT**;

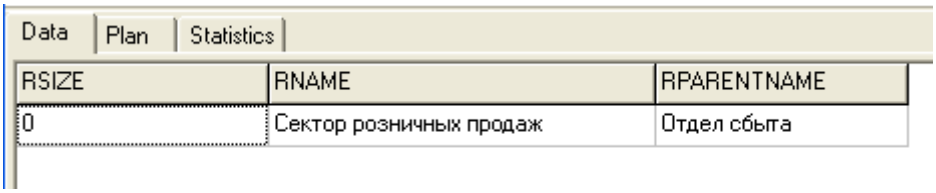
   EXECUTE PROCEDURE CountChiefNames

   Что равнозначно оператору: **SELECT * FROM CountChiefNames**
5. Рассмотрим несколько вариантов результата, получаемого после выполнения хранимой процедурой **Deplnfo**.
6. Для оператора **EXECUTE PROCEDURE DEPINFO(1)** результат показан на следующем рисунке.



Как видим, выходной параметр **rParentName** содержит значение **NULL**, поскольку подразделение **«Общее руководство»** находится на вершине иерархической структуры предприятия.

7. Теперь выполним оператор **EXECUTE PROCEDURE DEPINFO(6)**. Результат на следующем рисунке.



8. С помощью оператора **SELECT** можно выбрать только название и количество сотрудников в указанном подразделении:

   SELECT rName AS "Название", rSize AS "Штат"

   FROM Deplnfo(6)

Задание 6. Возвращение набора строк

Методические указания по выполнению задания:

1. До сих пор мы рассматривали примеры хранимых процедур, возвращающих только одну строку. Если процедура должна возвращать набор строк, то в ней используется конструкция **FOR SELECT ... DO** следующего вида:

   FOR оператор_5ELECT

   DO блок_операторов

2. Блок операторов после слова **DO** выполняется для каждой строки в наборе данных, который получают с помощью оператора **SELECT**, указанного после слова **FOR**. Если этот блок состоит из более чем одного оператора, то он должен быть ограничен словами **BEGIN** и **END**.
3. Создадим процедуру формирования списка подчиненных подразделений для подразделения, идентификатор которого передается в качестве входного параметра:

```
SET TERM ^ ;
CREATE PROCEDURE ChiIdDeptsList (pDepID INTEGER)
RETURNS (rID INTEGER, rName VARCHAR(100))
AS BEGIN
FOR SELECT "DeptID", "DeptFullName" FROM DEPS
WHERE "ParentDeptiD" = :pDepID
INTO :rID, :rName
DO SUSPEND;
END
```

В столбце **ParentDeptID** таблицы **DEPS** указывается идентификатор «родительского» подразделения. Если такого подразделения не существует (подразделение находится на вершине иерархической структуры предприятия), то в поле **ParentDeptID** содержится значение 0.

4. В процедуре **ChildDeptsList** выполняется оператор **SELECT**, возвращающий список подразделений, подчиненных текущему. Затем для каждой строки этого списка выполняется команда **SUSPEND**, которая передает в вызывающую программу данные, хранимые в параметрах **rID** и **rNAME**.
5. Для подобных хранимых процедур результат вызова с помощью операторов **EXECUTE PROCEDURE** и **SELECT** различен. Например, если выполнить оператор **EXECUTE PROCEDURE ChildDeptsList(5)**, то будет возвращена только одна (первая) строка результирующего набора данных. Для того чтобы получить весь набор, следует воспользоваться оператором **SELECT * FROM ChildDeptsList(5)**.

Задание 7. Использование переменных

Методические указания по выполнению задания:

1. Внутри хранимых процедур можно объявлять локальные переменные, которые затем используются в выражениях или для хранения промежуточных результатов запроса. Для того чтобы объявить переменную, применяют конструкцию **DECLARE VARIABLE**:

   CREATE PROCEDURE ...
   AS
   DECLARE VARIABLE имя__переменной1 тип_данных;
   DECLARE variable имя_переменной2 тип_даниых;
   DECLARE VARIABLE имя_переменноN тип_данных;
   begin
   ……..
   END
2. Если переменные в теле процедуры используются в правой части оператора присваивания, то доступ к ним осуществляется так же, как и к параметрам: через двоеточие. Если же переменной присваивается некоторое значение, то при обращении к ней двоеточие не указывают.

3. В качестве примера создадим процедуру **ExDepInfo**, которая возвращает информацию обо всех подразделениях в соответствии со следующим шаблоном:

Размер подразделения «Название_подразделения»

составляет: количество_сотрудников

В этой процедуре вначале выбираются все идентификаторы подразделений из таблицы **DEPS**, а потом для каждого из них извлекаются данные с помощью хранимой процедуры **DEPNFO**:

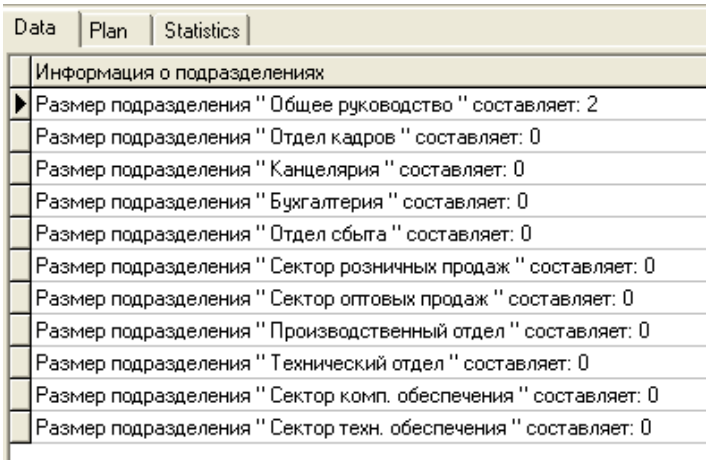
```
SET TERM ^ ;
CREATE PROCEDURE ExDepInfo
RETURNS (rlnfo VARCHAR(300) )
AS
DECLARE VARIABLE vDeptID INTEGER;
BEGIN
FOR SELECT “DeptID” FROM DEPS INTO :vDeptID DO
BEGIN
SELECT 'Размер подразделения " ' ||
rName || ' " составляет: ' ||
CAST(rSize AS VARCHAR(10))
FROM DepInfo(:vDeptID)
INTO :rlnfo;
SUSPEND;
END
END
```

Переменная **vDeptID** хранит идентификатор подразделения извлекаемый в цикле **FOR ... DO**. Для каждой строки, полученной в результате первого запроса SELECT, вызывается хранимая процедура **DepInfo**. Данные о названии (**rName**) и размере (**rSize**) подразделения, возвращаемые этой процедурой, используются для формирования результирующей строки **rInfo**. При этом строковые значения соединяются в одну строку с помощью оператора конкатенации ||, а для приведения целочисленного значении **rSize** к строковому типу используется встроенная функция **CAST**. Каждая полученная таким образом строка возвращается в вызывающую программу по команде **SUSPEND**.

4. Теперь можно выполнить следующий запрос:

   SELECT rlnfo AS "Информация о подразделениях" FROM ExDepInfo

   Результат выполнения этого запроса показан на рисунке.



Data | Plan | Statistics

Информация о подразделениях
Размер подразделения '' Общее руководство '' составляет: 2
Размер подразделения '' Отдел кадров '' составляет: 0
Размер подразделения '' Канцелярия '' составляет: 0
Размер подразделения '' Бухгалтерия '' составляет: 0
Размер подразделения '' Отдел сбыта '' составляет: 0
Размер подразделения '' Сектор розничных продаж '' составляет: 0
Размер подразделения '' Сектор оптовых продаж '' составляет: 0
Размер подразделения '' Производственный отдел '' составляет: 0
Размер подразделения '' Технический отдел '' составляет: 0
Размер подразделения '' Сектор комп. обеспечения '' составляет: 0
Размер подразделения '' Сектор техн. обеспечения '' составляет: 0

5. Применение конструкции ветвления **IF. ..THEN. . .ELSE** в SQL типично для языков программирования. Если условие, заданное в круглых скобках после ключевого слова IF, истинно (возвращает значение **TRUE**), то выполняется блок операторов, указанный после ключевого слова **THEN**. В противном случае выполняется блок операторов после ключевого слова **ELSE** (если оно присутствует). В качестве примера создадим хранимую процедуру **EmptyDeps**, возвращающую список подразделений, для которых не найдено ни одного сотрудника:

```
SET TERM ^ ;
CREATE PROCEDURE EmptyDepS
RETURNS (rDep VARCHAR(100))
AS
DECLARE VARIABLE vDeptID INTEGER;
DECLARE VARIABLE vDeptSize INTEGER;
BEGIN
FOR SELECT “DeptID” FROM DEPS
INTO :VDeptID DO
BEGIN
SELECT rSize, rName FROM Deplnfo(:vDeptID)
INTO :vDeptSize, :rDep;
IF (:vDeptSize = 0) THEN SUSPEND;
END
```

END

6. Здесь, в первом операторе **SELECT**, формируется список идентификаторов подразделений. Затем для каждого идентификатора (переменная **vDeptlD**) в цикле вызывается хранимая процедура **DepInfo**, с помощью которой для текущего подразделения извлекаются данные о количестве сотрудников (переменная **vDeptSize**) и названии (выходной параметр **rDep**). Если количество сотрудников равно нулю, то значение выходного параметра **rDep** возвращается в вызывающую программу.

   Вызовем эту процедуру:

   SELECT rDep AS "Подразделения без сотрудников"
   FROM EmptyDeps
   ORDER BY rDep

7. В хранимых процедурах можно использовать циклическую конструкцию **WHILE... DO**:

   WHILE (условие) DO блок_ операторов

8. До тех пор пока истинно условие цикла, выполняется блок операторов, указанный после ключевого слова **DO**.
9. В качестве примера создадим хранимую процедуру **Factorial**, которая вычисляет факториал числа, переданного в качестве входного параметра:

   CREATE PROCEDURE Factorial (N INTEGER)
   RETURNS (F INTEGER)
   AS BEGIN
   F = 1;
   WHILE (:N > 0) DO
   BEGIN
   F = :F * :N;
   N = :N - 1;
   END
   END

10. Теперь оператор **EXECUTE PROCEDURE Factorial 4** вернет число **24.**

Задание 8. Изменение существующих хранимых процедур

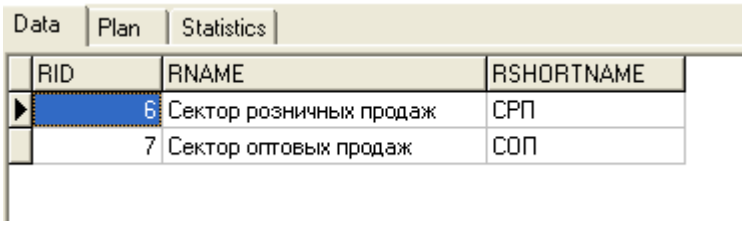
Методические указания по выполнению задания:

1. Для изменения входных параметров или тела существующей хранимой процедуры в

InterBase используется оператор **ALTER PROCEDURE**. Синтаксис этого оператора идентичен синтаксису оператора **CREATE PROCEDURE** за тем исключением, что вместо слова **CREATE** используется слово alter.

2. Изменим созданную ранее процедуру **ChildDeptsList** таким образом, чтобы она дополнительно возвращала сокращенные названия подчиненных подразделений:

```
SET TERM ^ ;
ALTER PROCEDURE CHIIDDEPTSLIST (pDepID INTEGER)
RETURNS (rID INTEGER, rName VARCHAR(100), rShortName VARCHAR(10))
AS BEGIN
FOR SELECT "DeptID", "DeptFullName", "DeptShortName" FROM DEPS
WHERE "ParentDeptiD" = :pDepID
INTO :rID, :rName, :rShortName
DO SUSPEND;
END
```



3. В тот момент, когда какая-либо хранимая процедура используется кем-то из пользователей базы данных, изменить ее невозможно.
4. Для удаления хранимых процедур используется оператор **DROP PROCEDURE**:

```
DROP PROCEDURE имя_працедуры
```

5. Процедуру, которая задействована в других хранимых процедурах или представлениях, удалить невозможно. То же самое относится и к процедурам, которые в момент выполнения оператора **DROP PROCEDURE** используются кем-то из пользователей базы данных.

Внеаудиторная самостоятельная работа:

Составить опорный конспект письменно в тетради по основным командам создания хранимых процедур.

Практическая работа №21

Создание генератора и триггеров. Каскадные воздействия

Цель работы: сформировать умения по выполнению команд создания генератора и триггеров в **InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**.

Задание 1. Создание триггеров. Оператор CREATE TRIGGER

Методические указания по выполнению задания:

1. **Триггер** — это особая хранимая процедура, которая вызывается в момент выполнения определенной операции над конкретной таблицей. Триггеры ассоциируются с такими операциями, как вставка (**INSERT**), обновление (**UPDATE**) и удаление (**DELETE**) данных.
2. Для создания триггера предназначен оператор **CREATE TRIGGER**, который имеет следующий синтаксис:

   CREATE TRIGGER имя_триггера FOR имя_таблицы
   индикатор_момента_срабатывания индикатор_операции
   POSITION приоритетность

AS опрелеления_переменных

BEGIN

операторы_ триггера

END

В качестве индикатора момента срабатывания используют ключевое слово **BEFORE** или **AFTER**. Ключевое слово **BEFORE** указывает на то, что триггер вызывается перед выполнением соответствующей операции (**INSERT, UPDATE** или **DELETE**). При создании триггеров, выполняемых после некоторой операции, вместо **BEFORE** используют ключевое слово **AFTER.**

Приоритетность, указываемая после ключевого слова **POSITION**, определяет порядок выполнения в том случае, если для одной и той же операции назначено несколько триггеров. Это значение должно находиться в диапазоне от 0 до 32767. При этом чем меньше значение, тем выше приоритетность триггера. Значение по умолчанию — 0. Если с некоторой операцией связан только один триггер, приоритетность для него можно не указывать.

Если несколько триггеров, созданных для одной и той же операции, имеют одинаковый приоритет, то они будут выполняться в случайном порядке.

3. В качестве примера создадим триггер, удаляющий из таблиц **FAMILY** и **JOBS** все строки, соответствующие записи, удаляемой из таблицы **STAFF**:

   CREATE TRIGGER STAFF_Delete FOR STAFF BEFORE DELETE AS BEGIN

   DELETE FROM FAMILY WHERE “ID” = OLD.ID;

   DELETE FROM JOBS WHERE “ID” = OLD.ID; END

   Обратите внимание на использование встроенной переменной **OLD**. Эта переменная служит для ссылки на значения, которые имели поля изменяемой или удаляемой строки перед выполнением операции **UPDATE** или **DELETE**. Существует еще одна встроенная переменная **NEW,** которая служит для ссылки на новые значения, помещаемые в строку командами **INSERT** или **UPDATE.**

4. В нашем примере данные в таблицах **FAMILY** и **JOBS** существуют одновременно только для сотрудника с идентификатором 1 (т.е. для Ивана Ивановича Иванова). Выполним следующий SQL-оператор:

   DELETE FROM STAFF WHERE ID = 1

   В результате данные будут автоматически удалены также из таблиц **FAMILY** и **JOBS.**

Задание 2. Взаимодействие с генераторами

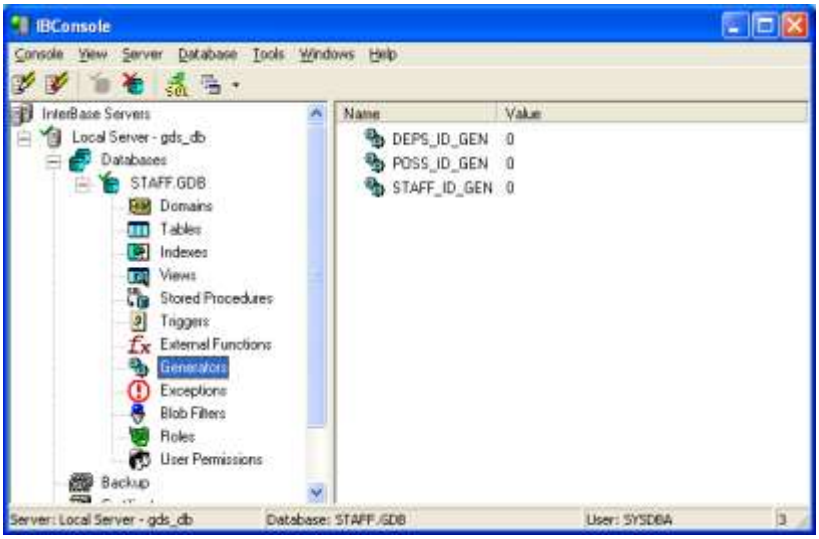
Методические указания по выполнению задания:

1. В терминах InterBase **генератор** — это последовательно возрастающее число, которое может автоматически вставляться в столбец при помощи встроенной функции **GEH_ID()**. Генераторы часто используются для формирования уникальных значений столбцов, входящих в первичный ключ. База данных InterBase может содержать любое количество генераторов, и они могут использоваться или обновляться в любой транзакции.
2. Создадим в базе данных **STAFF.gdb** три генератора, которые будут использоваться для внесения значений в первый столбец (столбец идентификатора) таблиц **STAFF**, **DEPS** и **POSS**. Для этого выполним следующие команды:

   CREATE GENERATOR STAFF_ID_GEN;

   CREATE GENERATOR DEPS_ID_GEN;

   CREATE GENERATOR POSS_ID_GEN
3. В результате в базе данных будут созданы три генератора. Для того чтобы убедиться в этом, в иерархической структуре базы данных в программе **IBConsole** необходимо выбрать элемент **Generators**.



После того как генераторы созданы, необходимо каким-то образом увеличивать их значение при добавлении новых строк в соответствующие таблицы. Для этого в InterBase используется встроенная функция **GEN_ID()**, которая вызывается внутри триггера при вставке новой строки в таблицу.

4. Создадим для таблиц **STAFF**, **DEPS** и **POSS** три триггера, вызывающих функцию **GEN_ID** для увеличения текущего значения соответствующего генератора:

```
SET TERM ^ ;
CREATE TRIGGER STAFF_Insert FOR STAFF
BEFORE INSERT POSITION 0
AS BEGIN
NEW.ID = GEN_ID(STAFF_ID_GEN, 1);
END^
CREATE TRIGGER DEPS_Insert FOR DEPS
BEFORE INSERT POSITION 0
AS BEGIN
NEW.DeptID = GEN_ID(DEPS_ID_GEN, 1);
END^
CREATE TRIGGER POSS_Insert FOR POSS
BEFORE INSERT POSITION 0
AS BEGIN
NEW.PosID = GEN_ID(POSS_ID_GEN, 1) ;
END^
SET TERM ;^
```

Для триггеров, выполняющих функцию **GEN_ID**, рекомендуется устанавливать наивысший номер с помощью ключевого слова **POSITION**. Установка **POSITION 0** означает, что триггер выполняется первым среди всех триггеров, связанных с операцией **INSERT** для данной таблицы.

В каждом из трех представленных выше триггеров в поле идентификатора добавляемой строки записывается значение соответствующего генератора, увеличенное на единицу. В качестве второго параметра функции **GEN_ID** можно также передавать числа больше единицы.

Задания 3. Установка начального значения генератора

Методические указания по выполнению задания:

1. Текущим значением для всех трех генераторов является 0. Это могло быть корректно в случае пустых таблиц **STAFF**, **DEPS** и **POSS**. Тогда при вставке первой записи в одну

из этих таблиц значение соответствующего генератора увеличилось бы на единицу, т.е. стало бы равным 1. Однако в нашем случае все три таблицы уже заполнены строками. Так, для таблицы **STAFF** наибольшее значение в уникальном ключевом поле составляет **27**, для таблицы **DEPS — 11**, а для таблицы **POSS — 21.**

Таким образом, прежде чем добавлять строки в таблицы **STAFF**, **DEPS** и **POSS**, необходимо установить новые начальные значения для соответствующих генераторов. Для этой цели в InterBase служит команда **SET GENERATOR**. Ее синтаксис:

SET GENERATOR имя_генератора TO исходное_значение

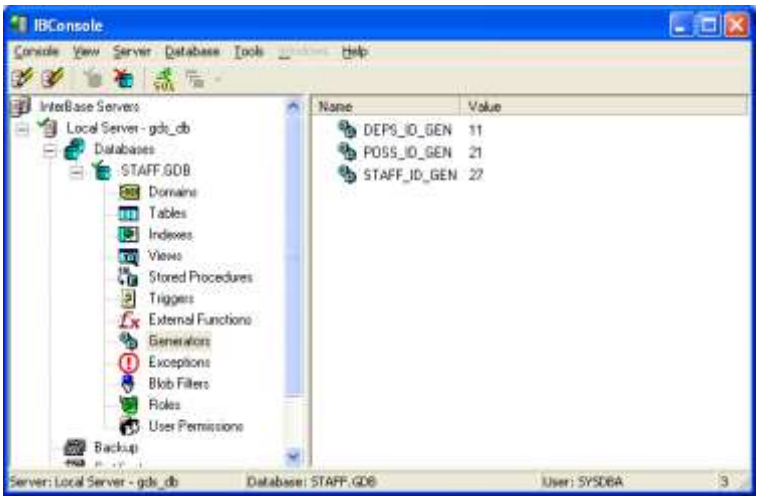
2. Выполним следующие SQL- операторы:

   SET GENERATOR STAFF_ID_GEN TO 27;

   SET GENERATOR DEPS_ID_GEN TO 11;

   SET GENERATOR POSS_ID_GEN TO 21

   Соответствующие изменения отобразятся в программе **IBConsole.**



3. В качестве примера добавим в таблицу **STAFF** ранее удаленную строку с данными о генеральном директоре Иване Ивановиче Иванове:

   INSERT INTO STAFF ("ID", "LastName", "FirstName", "FatherName", "Zip", "IdCode","TabNum", "BirthDate", "DepID", "PosID", "Salary")

   VALUES(0, 'Иванов', 'Иван', 'Иванович', 2001, '111', '111', '03.05.1966', 1, 1, 100560)

   Для полей, связанных с генераторами, значение, указанное в списке **VALUES**, роли не играет. Главное, чтобы оно не было равно **NULL**. В результате в таблицу **STAFF** была добавлена новая строка, у которой в поле **ID** указано значение **28**. Это же значение стало текущим для генератора **STAFF_ID_GEN.**

Задание 4. Удаление генераторов

1. Прямой SQL-команды удаления генераторов в InterBase не существует, поэтому для удаления какого-либо генератора из базы данных используют своего рода обходной маневр.
2. Ссылки на создаваемые генераторы заносятся в служебную таблицу **RBB$GEHERATORS**. Следовательно, для того чтобы удалить какой-либо генератор, например **POSS_ID_GEN**, достаточно выполнить такую SQL-команду:

   DELETE FROM RDB$GENERATOR
   WHERE RDB$GENERATOR_NAME = 'POSS_ID_GEN'

Задание 5. Взаимодействие с исключениями

Методические указания по выполнению задания:

1. В терминах **InterBase исключение** – это механизм сообщении об ошибках, определенных пользователем. Этот механизм реализуется с помощью хранимых процедур и триггеров. Для определения нового исключения в **InterBase** используется команда **CREATE EXCEPTION**.
2. Создадим три исключения, которые возникают в тот момент, когда кто-то пытается удалить из таблицы **DEPS, POSS** или **REGIONS** строки, ключевые значения которых используются в таблице **STAFF**. Подобное удаление в любом случае невозможно выполнить, поскольку ранее с помощью внешних ключей с этими таблицами были связаны соответствующие столбцы таблицы **STAFF**. Тем не менее при выполнении некорректной операции исключения позволяют отобразить пользователю удобочитаемое сообщение.

   CREATE EXCEPTION Dept_IS_Busy 'На это подразделение есть ссылка в таблице сотрудников. Удаление невозможно.';

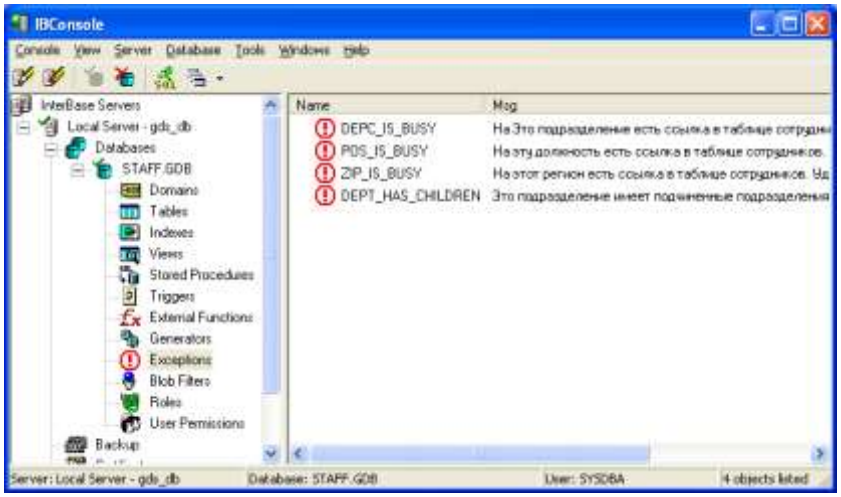
   CREATE EXCEPTION Pos_Is_Busy 'На эту должность есть ссылка в таблице сотрудников. Удаление невозможно.';

   CREATE EXCEPTION Zip_Is_Busy 'На этот регион есть ссылка в таблице сотрудников. Удаление невозможно.'
3. Создадим еще одно исключение, возникающее при удалении подразделения, которое имеет подчиненные подразделения:

   CREATE EXCEPTION Dept_Has_Children 'Это подразделение имеет подчиненные

подразделения.'

Созданные исключения можно просмотреть в программе **IBConsole**, выделив в иерархической структуре базы данных **STAFF.gdb** элемент **Exceptions**.



4. Для вызова исключений используется оператор вида: **EXCEPTION имя_исключения.**

   Создадим триггеры для вызова созданных нами исключений;

   SET TERM ^;

   CREATE TRIGGER DEPS_Delete FOR DEPS

   BEFORE DELETE

   AS

   DECLARE VARIABLE CountOfRefs integer;

   BEGIN

   SELECT COUNT("ID") FROM STAFF WHERE "DepID" = OLD. "DeptID"

   INTO :CountOfRefs;

   IF (:CountOfRefs > 0) THEN EXCEPTION Dept_Is_Busy;

   END^

   CREATE TRIGGER POSS_Delete FOR POSS

   BEFORE DELETE

   AS

   DECLARE VARIABLE CountOfRefs integer;

   BEGIN

   SELECT COUNT("ID") FROM STAFF WHERE "PosID"=OLD."PosID"

   INTO :CountOfRefs;

   IF (:CountOfRefs > 0) THEN EXCEPTION Pos_IS_Busy;

END^

CREATE TRIGGER REGION_Delete FOR REGIONS

BEFORE DELETE

AS

DECLARE VARIABLE CountOfRefs integer;

BEGIN

SELECT COUNT("ID") FROM STAFF WHERE "Zip" = OLD."Zip"

INTO :CountOfRefs;

IF (:CountOfRefs > 0) THEN EXCEPTION Zip_IS_Busy;

END^

SET TERM ;^

5. Теперь попытаемся удалить из таблицы **DEPS** строку, ссылка на которую присутствует в таблице **STAFF**:

   DELETE FROM DEPS WHERE "DeptID" = 1

   В результате будет получено сообщение об ошибке, представленное на рисунке.

6. Попытаемся также удалить из таблицы **DEPS** строку, на которую нет ссылок в таблице **STAFF**, однако с ней связаны подчиненные подразделения:

   DELETE FROM DEPS WHERE DeptID = 9

Задание 6. Изменение триггеров. Оператор ALTER TRIGGER

Методические указания по выполнению задания:

1. Для изменения существующего триггера в InterBase предназначен оператор **ALTER TRIGGER**. Его синтаксис:

   ALTER TRIGGER имя_триггера

   индикатор_активности

   индикатор_момента_срабатывания

   индикатор_операции

   POSITION приоритетность

   AS BEGIN

тело_триггера

END

Этот оператор почти идентичен оператору **CREATE TRIGGER** за тем исключением, что не указывается имя таблицы, а также можно временно отключить или вновь активизировать триггер с помощью индикатора **INACTIVE** или **ACTIVE.**

2. Если необходимо временно отключить автоматическую вставку строки в таблицу **JOBS** после добавления информации о новом сотруднике в таблицу **STAFF**, можно выполнить следующий оператор: **ALTER TRIGGER STAFF_Insert_Job INACTIVE**

   Для повторной активизации триггера **STAFF_Insert_Job** используется оператор: **ALTER TRIGGER STAFF_Insert_Job ACTIVE**
3. Если при обновлении триггера какие-либо аргументы пропущены, то их значения принимаются равными значениям, которые были определены в момент выполнения оператора **CREATE TRIGGER** или последнего оператора **ALTER TRIGGER** для текущего триггера.
4. Для удаления триггера из базы данных используется оператор вида:

   DROP TRIGGER имя_триггера
5. При этом могут быть удалены только те триггеры, которые не задействованы в активной транзакции.

Внеаудиторная самостоятельная работа:

Составить письменно в тетради опорный конспект по основным командам создания генератора и триггеров.

Практическая работа №22

Сортировка, поиск и фильтрация данных в базах данных и выборках.

Цель работы: Ознакомление с механизмами поиска данных, фильтрации записей и использование индексов для сортировки.

Последовательный перебор

В программах, работающих с базами данных, часто используют *поиск* данных. Для чего еще нужны *базы данных*, как не для этого? Самый простой, но в то же время и самый медленный, "тяжеловесный" *поиск*, это, пожалуй, ***последовательный перебор***. Вы переходите на первую *запись* таблицы, создаете цикл, который длится до последней записи, и внутри этого *цикла* проверяете необходимое условие. Также можно делать и *обратный* перебор, от последней записи к первой. В таблице 1 приведены все свойства и

методы наборов данных (TTable/ADOTable, TQuery/ADOQuery), которые могут быть использованы при организации *последовательного перебора*:

Таблица 1. Свойства и методы набора данных, которые могут быть задействованы при *последовательном переборе*

Свойства и методы	**Описание**
Eof	Свойство логического типа. Принимает значение True, если достигнут конец таблицы, или если таблица пуста, и Falseв противном случае.
Bof	Свойство логического типа. Принимает значение True, если достигнуто начало таблицы, и False в противном случае.
Next	Метод. Делает текущей следующую запись набора данных.
Prior	Метод. Делает текущей предыдущую запись набора данных.
First	Метод. Делает текущей первую запись набора данных.
Last	Метод. Делает текущей последнюю запись набора данных.

Пример:

```
//перешли на первую запись:
fDM.TLichData.First;
//делать, пока не конец таблицы:
while not fDM.TLichData.Eof do begin
  if fDM.TLichData['Фамилия'] = 'Иванов' then
     break;  //нашли нужную запись, и вышли из цикла
  fDM.TLichData.Next; //иначе перешли на следующую запись
end; //while
```

Как видно из примера, мы делаем *прямой последовательный перебор* от первой записи до последней. Получить или изменить *значение*нужного поля можно, указав имя поля в квадратных скобках после имени набора данных. Например:

```
Edit1.Text := fDM.TLichData['Фамилия']; //получили значение
fDM.TLichData['Фамилия']:= Edit1.Text; //изменили значение
```

Приведенный пример поиска нужной записи допустим, если в таблице имеется не более сотни-другой записей, а условная проверка достаточно сложна. Но обычно программисты этот способ не используют, или используют только в крайнем случае. Далее рассмотрим другие способы поиска.

Метод Locate

Метод ***Locate*** ищет первую *запись*, удовлетворяющую *условию поиска*. Если *запись* найдена, метод делает ее текущей и возвращает True. В противном случае метод возвращает False и *курсор* не меняет положения. *Поле, по* которому ведется *поиск*, не обязательно должно быть индексировано. Однако если *поле* индексировано, то метод ищет *запись по* индексу, что значительно ускоряет *поиск*. *Поиск* может вестись как *по* одному полю, так и *по* нескольким полям. Метод имеет три параметра:

```
function Locate (const KeyFields: String; const KeyValues: Variant;
                 Options: TLocateOptions) : Boolean;
```

Параметр KeyFields задает *поле* или *список* полей, *по* которым ведется *поиск*. Если имеется несколько полей, их разделяют точкой с запятой.

Параметр KeyValues является вариантным массивом, в котором задаются *критерии поиска*. При этом первое *значение* KeyValues ставится в соответствие с первым полем, указанным в KeyFields. Второе - со вторым, и так далее.

Третий *параметр* Options позволяет задать некоторые опции поиска:

- loCaseInsensitive - поиск ведется без учета высоты букв, то есть, считаются одинаковыми строки "строка", "Строка" или "СТРОКА".
- loPartialKey - запись будет удовлетворять условию, если ее часть содержит искомый текст. То есть, если мы ищем "ст", то удовлетворять условию будут "строка", "станция", "стажер" и т.п.
- Пустой набор **[]** указывает, что настройки поиска игнорируются. То есть, строка ищется "как есть".

Примеры использования метода *Locate*:

```
Table1.Locate('Фамилия', Edit1.Text, []);
Table1.Locate('Фамилия;Имя',
  VarArrayOf(['Иванов', 'Иван']), [loCaseInsensitive]);
```

Как видно из примера, если для поиска вы используете одно *поле*, то *значение* может передаваться напрямую из компонента Edit. Если же вы используете *список* полей, то должны передать в метод *массив* вариантов, в которых содержатся искомые значения, *по* одному на каждое *поле*. При установке компонента ADOTable в раздел uses прописывается *модуль* **ADODB**, который содержит описания всех свойств, методов и событий компонента. Желательно использовать метод в том модуле, где установлен этот *компонент*.

Рассмотрим применение этого метода на примере. Откройте проект. Перейдите на *модуль* DM, где у нас хранятся компоненты доступа к базе данных. Процедуру поиска реализуем в этом модуле, а чтобы с ней можно было работать из других форм, опишем ее в разделе public:

```
public
  { Public declarations }
  procedure MyLocate(s: String);
```

Как видите, в процедуру передается *параметр* - строка. В ней мы будем передавать искомую фамилию. Если *курсор* находится на описании нашей процедуры, то нажмите <Ctrl + *Shift* + C>, чтобы сгенерировать процедуру автоматически. Процедура будет иметь следующий код:

```
procedure TfDM.MyLocate(s: String);
begin
  TLichData.Locate('Фамилия', s, [loPartialKey]);
end;
```

Таким образом, при нахождении подходящей записи *курсор* будет перемещаться к ней.

На главной форме выделите *компонент* Edit, предназначенный для поиска *по* фамилии. Создайте для него событие *onChange*, которое наступает при изменении текста в *поле* компонента. В *созданной процедуре* пропишите вызов поиска:

```
fDM.MyLocate(Edit1.Text);
```

Сохраните пример, скомпилируйте и опробуйте результаты поиска. Метод *Locate* рекомендуется использовать везде, где это возможно, поскольку он всегда пытается применить наиболее быстрый *поиск*. Если *поле* индексировано, и использование индекса ускорит процесс поиска, *Locate* использует *индекс*. Если *поле* не имеет индекса, *Locate* все равно ищет данные наиболее быстрым способом. Это делает вашу программу независимой от индексов.

Метод Lookup

Метод ***Lookup***, в отличие от *Locate*, не меняет положение курсора в таблице. Вместо этого он возвращает значения некоторых ее полей. Причем в отличие от *Locate*, этот метод осуществляет *поиск* лишь на точное соответствие. Такой способ поиска востребован реже,

однако в иных случаях этим методом очень удобно пользоваться. Рассмотрим *синтаксис* этого метода.

```
function Lookup (const KeyFields: String;
                 const KeyValues: Variant;
                 const ResultFields: String) : Variant;
```

Как вы видите, первые два параметра такие же, как у *Locate*. А вот третий *параметр* и возвращаемое *значение* отличаются. В строке ResultFields через точку с запятой перечисляются поля таблицы, значения которых метод должен вернуть. Возвращаются эти значения в виде вариантного массива. Проблема в том, что вернуться может *значение* Null, то есть, ничего, или Empty (пустой) и это нужно проверять. Рассмотрим работу метода *Lookup* на примере нашей программы.

Прежде всего, вспомним, как работает *тип данных* Variant. В переменную типа Variant можно поместить любое *значение*, в том числе и *массив*. Этот *тип данных* обычно используют, когда не известно заранее, данные какого типа нам понадобятся на этапе выполнения программы. Когда переменной типа Variant присвоено *значение*, имеется возможность проверить *тип данных* этого значения. Для этого служит *функция**VarType()*:

```
function VarType(const V: Variant): TVarType;
```

В качестве параметра в функцию передается *переменная* вариантного типа. *Функция* возвращает *значение* типа TVarType. Это *значение*указывает, какого типа данные содержатся в переменной. *Значение* может быть varSmallint (короткое целое), varInteger (целое), varCurrency (денежный формат) и так далее. Чтобы увидеть полный *список* возвращаемых функцией значений, в редакторе кода установите *курсор* на название функции и нажмите <Ctrl + F1>, вызвав контекстный справочник. Нас же в данном примере интересуют всего два значения: varNull (записи нет) и varEmpty (*запись* пустая). Если в программе мы заранее не проведем проверку на эти значения, то вполне можем вызвать ошибку программы. Если же *поиск* прошел успешно, то будет возвращен *массив* вариантных значений, элементы которого начинаются с нуля. Каждый элемент массива будет содержать данные одного из указанных полей.

Загрузите проект программы. Для поиска воспользуемся кнопкой с надписью "Найти", расположенной в верхней части главной формы. Идея такова: *пользователь* вводит в *поле* Edit1 какую то фамилию и нажимает кнопку "Найти". Событие onClick этой кнопки собирает в строковую переменную значения четырех указанных полей найденной записи. Причем после каждого значения в строку добавляется символ "#13" (переход на новую строку), формируя многострочный отчет. Затем эту строку мы выведем на экран функцией ShowMessage().

Итак, в окне главной формы дважды щелкните *по* кнопке "Найти", генерируя событие onClick. Полный листинг процедуры приведен ниже:

```
{щелкнули по кнопке Найти}
procedure TfMain.BitBtn1Click(Sender: TObject);
var
  myLookup: Variant; //для получения результата
  s : String; //для отчета
begin
  //получаем результат:
  myLookup := fDM.TLichData.Lookup('Фамилия', Edit1.Text,
          'Фамилия;Имя;Отчество;Образование');
  //проверяем, не Null ли это:
```

```
  if VarType(myLookup) = varNull then
    ShowMessage('Сотрудник с такой фамилией не найден!')
  else if VarType(myLookup) = varEmpty then
    ShowMessage('Запись не найдена!')
  //если это массив, то из его элементов собираем
  //многострочную строку:
  else if VarIsArray(myLookup) then begin
    s := myLookup[0] + #13 +  myLookup[1] + #13 +
         myLookup[2] + #13 + myLookup[3];
    //и выводим ее на экран:
    ShowMessage(s);
  end; //else if
end;
```

Комментарии достаточно подробны, чтобы вы разобрались с кодом. Сохраните проект, скомпилируйте его и запустите. Опробуйте этот способ поиска.

Фильтрация данных

Фильтрацию данных применяют не реже а, пожалуй, даже чаще, чем *поиск*. Разница в том, что при поиске данных *пользователь* видит все записи таблицы, при этом *курсор* либо переходит к искомой записи, либо он получает данные этой записи в виде результата работы функции. При фильтрации дело обстоит иначе. *Пользователь* в результате видит только те записи, которые удовлетворяют условиям фильтра, остальные записи становятся скрытыми. Конечно, таким образом искать нужные данные проще. Можно указать в условиях фильтра, что требуется вывести всех сотрудников, чья фамилия начинается на "И". *Пользователь* увидит только их. А можно и *по*-другому: вывести всех сотрудников, которые поступили на работу в период между 2000 и 2005 годом. Короче говоря, удобство работы пользователя с вашей программой зависит от вашей фантазии. Рассмотрим основные способы фильтрации записей.

Свойство Filter

Свойство ***Filter*** - наиболее часто используемый способ фильтрации записей, имеет тип String. Вначале программист задает условия фильтрации в этом свойстве, затем присваивает логическому свойству *Filtered значение* True, после чего *таблица* будет отфильтрована. Условия фильтрации должны входить в строку, например:

```
fDM.TLichData.Filter := 'Фамилия ="Иванов"';
```

По правилам синтаксиса, если внутри строки встречается *апостроф*, его нужно дублировать. Приведенный выше пример в результате содержит условие:

```
Фамилия = 'Иванов'
```

Применяя это свойство, достаточно сложных условий задать невозможно, но если условия фильтрации просты, то данный способ незаменим. Опробуем фильтрацию записей на примере нашего приложения. Откройте событие *onChange* компонента Edit, изменим его немного. Закомментируйте или удалите вызов процедуры поиска MyLocate, и впишите следующий код:

```
//fDM.MyLocate(Edit1.Text); - закомментировали

  fDM.TLichData.Filter := 'Фамилия >=' + QuotedStr(Edit1.Text);
  fDM.TLichData.Filtered := True;
```

Откомпилируйте проект и запустите его на выполнение. При введении только первой буквы фамилии записи уже начинают фильтроваться. К примеру, если мы ввели букву "Л", то остаются записи с фамилиями, начинающимися от буквы "Л" до конца алфавита. Можно

также улучшить *поиск*, если при этом еще отсортировать записи *по* индексу, но об этом чуть позже. *Функция* QuotedStr() возвращает переданный ей текст, заключенный в апострофы. Условие фильтра можно было бы описать и так:

```
fDM.TLichData.Filter := 'Фамилия >=''' +
    Edit1.Text + '''';
```

Сложность заключается в том, что в этом случае приходится считать апострофы. *Функция* QuotedStr() помогает решить эту проблему.

Событие onFilterRecord

Это событие возникает при установке значения True в свойстве *Filtered*. Применение этого способа имеет большой плюс, и большой минус. Плюс в том что, сгенерировав это событие, программист получает возможность задать гораздо более сложные условия фильтрации. Минус же заключается в том, что проверка осуществляется перебором всех записей таблицы. Если *таблица* содержит очень много записей, процесс фильтрации может затянуться.

В событие передаются два параметра. Первый *параметр* - набор данных DataSet. С ним можно обращаться, как с именем фильтруемой таблицы. Второй *параметр* - *логическая переменная* Accept. Этой переменной нужно передавать результат проверки условия фильтра. Если *переменная* Accept возвращает False, то *запись* не принимается, и не будет отображаться. Соответственно, если возвращается True, то *запись* принимается. Рассмотрим этот способ на примере. Суть примера в следующем: необходимо отфильтровать записи *по* начальным (или всем) буквам фамилии, вводимым пользователем в *поле* Edit1. В предыдущем примере, если бы мы ввели букву "И", то вышли бы фамилии, первой буквой которых были бы "И" - "Я". Это не так удобно. Сделаем так, чтобы если *пользователь* введет букву "И", то останутся только фамилии, начинающиеся на "И". Если *пользователь* введет еще букву "в", то останутся только фамилии, начинающиеся на "Ив", и так далее. Поочередно вводя начальные буквы, *пользователь* доберется до нужных фамилий.

Для начала подготовим *модуль* данных. В нем нам потребуется создать глобальную переменную ed, чтобы мы могли передавать в нее текст из компонента Edit1:

```
var
  fDM: TfDM;
  ed: String; //текст из Edit1
```

Этого действия можно было бы избежать, если бы *компонент* ADOTable, подключенный к таблице LichData, располагался на главной форме. Но поскольку он находится в модуле данных, то и событие *onFilterRecord* будет сгенерировано в нем. А в этом событии нам нужно будет знать, что в данный момент находится в *поле* ввода Edit1. Именно для этого и нужна глобальная *переменная* ed.

Далее выделяем TLichData, то есть, *компонент* ADOTable, подключенный к таблице LichData. На вкладке Events (События) инспектора объектов найдите событие *onFilterRecord* и дважды щелкните *по* нему, сгенерировав процедуру. Полный листинг процедуры:

```
{onFilterRecord главной таблицы}
procedure TfDM.TLichDataFilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
var
  s : String; //для значения поля
begin
  //получаем столько начальных букв из поля Фамилия,
```

```
  //сколько букв имеется в переменной ed:
  s := Copy(DataSet['Фамилия'], 1, Length(ed));
  //делаем проверку на совпадение значений:
  Accept := s = ed;
end;
```
Здесь в переменную s попадает столько начальных букв из поля "Фамилия", сколько букв содержит в данный момент *компонент* Edit1 на главной форме (эти буквы мы передадим в переменную ed чуть позже). Если текст в переменной s совпадает с текстом из поля Edit1, то переменной Accept присваивается True, и *запись* принимается. Иначе *запись* отфильтровывается. Не забудьте сохранить проект.

Далее перейдем в главную форму. Нужно удалить весь текст из события *onChange* компонента Edit1, и вписать новый:

```
{Изменение поиска по фамилии}
procedure TfMain.Edit1Change(Sender: TObject);
begin
  //если в поле Edit1 есть хоть одна буква,
  if Edit1.Text <> '' then begin
    fDM.TLichData.Filtered := False; //отключаем фильтр
    ed := Edit1.Text; //передаем в fDM новый текст
    fDM.TLichData.Filtered := True; //включаем фильтр
  end
  //если букв нет, фильтрацию отключаем:
  else fDM.TLichData.Filtered := False;
end;
```

Вот и все. Что же тут у нас происходит? Как только *пользователь* введет хоть одну букву, срабатывает событие *onChange* компонента Edit1. Если в Edit1 есть хоть одна буква, то мы вначале отключаем фильтрацию, отменяя прошлый фильтр, если он был. Затем мы передаем в глобальную переменную ed, расположенную в модуле данных, текст из Edit1. Далее снова включаем фильтр. При этом срабатывает событие *onFilterRecord* нашей таблицы, и в этом событии сравнивается текущее *значение* переменной ed и записей поля "Фамилия".

Сохраните проект, скомпилируйте и запустите программу. Проверьте, как фильтруются записи. Имея воображение, в событии *onFilterRecord* можно устраивать сколь угодно сложные проверки. Ведь в этом событии можно сравнивать не одно *поле*, а несколько, причем поля не обязательно должны быть индексированы. Вы можете проверять на совпадение хоть все поля таблицы, и поскольку фильтрация происходит путем перебора записей, то усложнение условных проверок заметно не замедлит этот процесс.

Использование индексов

Создание индексных полей обеспечивает сортировку данных *по* этим полям, что также облегчает *поиск* данных - ведь найти нужную фамилию или имя проще, если они отсортированы *по* алфавиту. Причем имеется возможность сортировать записи не только *по* возрастанию, но и *по*убыванию, хотя в большинстве руководств *по* Delphi эта возможность не описывается.

При создании в базе данных таблицы LichData мы указали поля "Фамилия" и "Имя", как индексированные. Этим и воспользуемся. Чтобы включить сортировку записей *по* полю "Фамилия", достаточно указать название поля в свойстве ***IndexFieldNames*** таблицы:

```
fDM.TLichData.IndexFieldNames := 'Фамилия';
```

Если требуется отключить сортировку, этому свойству присваивается пустая строка:

fDM.TLichData.IndexFieldNames := '';
Существует еще одна хитрость, о которой мало где можно прочитать. При *индексировании таблицы* к имени поля можно прибавить строку " *ASC* ", если мы желаем сортировать в возрастающем порядке (*по* умолчанию), или " *DESC* ", если сортируем в убывающем порядке. *Сортировка*" *ASC* " используется *по* умолчанию. Добавим возможность сортировки *по* фамилии и имени в нашу программу. Для этого на главную форму установим *компонент* TPopupMenu с вкладки Standard *палитры компонентов*. Дважды щелкните *по* компоненту, чтобы открыть редактор *меню*. Создадим следующие пункты:
Сортировать по фамилии
Сортировать по имени
Не сортировать
-
Обратная сортировка
В редакторе *меню* выделите *пункт* "Сортировать *по* фамилии" и измените свойство Name этого пункта на NFam. *Пункт* "Сортировать *по*имени" переименуйте в NImya. *Пункт* "Не сортировать" - в NNet, а *пункт* "Обратная *сортировка*" - в NObrat.
Вначале создайте обработчик событий для пункта "Не сортировать" (дважды щелкните *по* пункту). Тут все просто:

```
{Не сортировать}
procedure TfMain.NNetClick(Sender: TObject);
begin
  fDM.TLichData.IndexFieldNames := '';
end;
```

Для обработчика событий пункта "Сортировать *по* фамилии" код немного сложней:

```
{Сортировать по фамилии}
procedure TfMain.NFamClick(Sender: TObject);
var
  stype : String;
begin
  //выбираем направление сортировки:
  if NObrat.Checked then stype := ' DESC' //обратная сортировка
  else stype := ' ASC';  //прямая сортировка
  //сортируем
  fDM.TLichData.IndexFieldNames := 'Фамилия' + stype;
end;
```

Здесь, в зависимости от состояния свойства Checked пункта "Обратная *сортировка*" мы присваиваем строковой переменной stype либо *значение* ' *ASC* ' (прямая *сортировка*), либо ' *DESC* ' (обратная *сортировка*). Обратите внимание, что первым символом строки является *пробел*, он нужен, чтобы строка не "прилепилась" к названию поля. Далее мы устанавливаем *индекс*, указывая имя поля и добавляя к нему *значение*переменной stype. Таким образом, если Checked пункта "Обратная *сортировка*" имеет *значение* True (галочка установлена), мы добавляем ' *DESC* ', или ' *ASC* ' в противном случае. В результате имя индексного поля может быть либо "Фамилия *ASC* ", либо "Фамилия *DESC* ".
Сортировку *по* имени кодируем аналогичным образом:

```
{Сортировать по имени}
procedure TfMain.NImyaClick(Sender: TObject);
var
  stype : String;
```

```
begin
  //выбираем направление сортировки:
  if NObrat.Checked then stype := ' DESC'
  else stype := ' ASC';
  //сортируем
  fDM.TLichData.IndexFieldNames := 'Имя' + stype;
end;
```

Нам осталось указать код пункта всплывающего *меню* "Обратная *сортировка*". Тут нам нужно не просто установить галочку, если ее не было, но также проверить - есть ли *сортировка по* какому либо полю? Если *таблица* отсортирована, требуется ее пересортировать *по* этому же полю, но уже в обратном порядке. Вот код:

```
{Команда "Обратная сортировка"}
procedure TfMain.NObratClick(Sender: TObject);
begin
  //изменяем направление сортировки
  NObrat.Checked := not NObrat.Checked;
  //если сортировка по фамилии, пересортируем
  if Pos('Фамилия',fDM.TLichData.IndexFieldNames)>0 then
    fMain.NFamClick(Sender);
  //если сортировка по имени, пересортируем
  if Pos('Имя',fDM.TLichData.IndexFieldNames)>0 then
    fMain.NImyaClick(Sender);
end;
```

Как видите, мы использовали функцию Pos(), которая возвратит ноль, если в строке не найдено указанной подстроки, или номер символа, с которого эта *подстрока* начинается, если она есть. Нам нужно определить, не входит ли в имя индексного поля "Фамилия" или "Имя". Ведь к имени поля добавлена строка ' *ASC* ' или ' *DESC* ', так что прямая проверка

```
if fDM.TLichData.IndexFieldNames = 'Фамилия' then
```

результата не даст, в любом случае результатом было бы False. Ну а для пересортировки мы вызываем соответствующий *пункт меню*, чтобы не писать код сортировки еще раз, например:

```
fMain.NFamClick(Sender);
```

Следует заметить, что при большом количестве записей в таблице смена индексного поля будет несколько замедлять работу приложения. Тем не менее, *индексация* таблицы - очень удобный и часто применяемый способ организации вывода записей.

В свойстве PopupMenu верхней сетки DBGrid1 выберите созданное только что всплывающее *меню*, чтобы оно открывалось только над этой сеткой, сохраните проект, скомпилируйте его и опробуйте сортировку данных.

Напоследок заметим, что мы имеем возможность применить одновременно и фильтрацию записей, и их индексацию. Это позволяет нам создать достаточно мощный и удобный для пользователя механизм поиска записей в нашей программе.

Практическая работа №23

Обработка транзакций. Кэширование изменений. Работа с транзакциями в InterBase.

Цель работы: сформировать умения по работе с транзакциями в **InterBase**; познакомится с особенностями многоверсионной архитектуры, уровнями изолированности транзакций.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

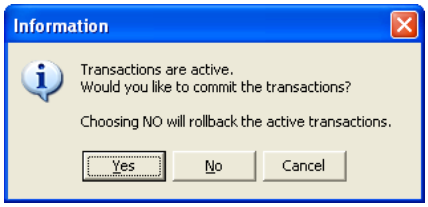
Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase.**

Теоретическая справка:

- **Транзакция** - это группа вносимых в базу данных изменений, которая рассматривается SQL-сервером как единое целое.
- Механизм транзакций реализован для устранения конфликтов при конкурентном доступе, а также для защиты от случайных потерь данных в результате сбоев. В одну и ту же базу данных могут одновременно вносить изменения множество пользователей, и каждый из них формирует собственную транзакцию. В конце концов, пользователь может или подтвердить все изменения, входящие в состав транзакции, или отменить их. Такой механизм удобен в случае сбоев, когда во время выполнения транзакций база данных оказывается в несогласованном состоянии, при котором часть изменений уже внесена, а часть нет. В подобной ситуации все частичные изменения, входящие в состав транзакции, могут быть отменены для возвращения базы данных в исходное согласованное состояние.
- Таким образом, возможны только два сценария: SQL-сервер фиксирует в базе данных

изменения, выполненные транзакцией; ни одно из изменении, входящих в состав транзакции, не фиксируется.

3. Для фиксации всех изменений, выполненных в пределах активной транзакции, используется SQL-команда **COMMIT**, а для их отмены — команда **ROLLBACK**. Действие обеих команд распространяется только на транзакции, которые имели место после выполнения последней команды **COMMIT**. Другими словами, изменения, уже сохраненные в базе данных, отменить нельзя.
4. В среде **InterBase** вопрос о выполнении команды **COMMIT** или **ROLLBACK** для активных транзакций появляется при попытке закрыть окно **Interactive SOL**.



Если в этом окне щелкнуть на кнопке **Yes**, то будет выполнена команда **COMMIT**, кнопке **No** соответствует команда **ROLLBACK**, а кнопке **Cancel** - отмена выхода из окна **Interactive SQL**.

5. Кроме того, находясь в окне **Interactive SQL**, транзакцию можно зафиксировать в базе данных или отменить с помощью соответствующей команды меню **Transactions**.
6. Если оператор **INSERT INTO, UPDATE** или **DELETE** выполняется в какой-либо программе управления базами данных, то транзакция начинается автоматически. Однако всеми SQL-серверами также поддерживаются специальные SQL-команды, выполняющие явную инициализацию новой транзакции командой **SET TRANSACTION**. При этом для нее можно определить характер взаимодействия с другими транзакциями, обрабатывающими те же данные. Так, по умолчанию установлено, что в случае обнаружения блокировки данных со стороны другой транзакции, текущая транзакция ожидает, пока эти данные не будут освобождены и только затем производит какие-либо операции. Если же необходимо, чтобы транзакция при обнаружении блокировки данных со стороны другой транзакции сразу же возвращала ошибку, используется атрибут **NO WAIT**:

SET TRANSACTION NO WAIT

7. Еще одна возможность, определенная для команды **SET TRANSACTION**, заключена в установке уровня разграничения с конкурирующими транзакциями. Для этой цели

служит атрибут **ISOLATION LEVEL**, после выбора которого, указывается одно из следующих значений:

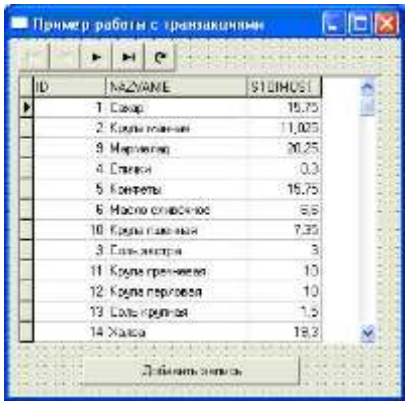
- **SNAPSHOT (значение по умолчанию)** — в момент инициализации транзакции выполняется чтение копии данных, над которой затем выполняются операции; изменения, внесенные в это время другими транзакциями, невидимы; доступ одной транзакции к промежуточным или окончательным результатам другой — исключен;
- **READ COMMITED** — изменения, зафиксированные для других конкурирующих транзакций, сразу же становятся видимыми; незафиксированные изменения со стороны конкурирующих транзакций — невидимы; если строка уже изменена другой транзакцией, она не может быть обновлена текущей транзакцией.

Задание:

Создайте приложение, работающее с таблицей **Tovar** базы данных. Приложение будет использовать две транзакции с различными параметрами - одну для чтения данных, другую для записи.

Методические указания по выполнению задания:

1. Прежде всего, убедитесь, что сервер **InterBase** запущен.
2. Далее, создайте в **Delphi** новый проект. На главную форму поместите **DBNavigator**, сетку **DBGrid** и простую кнопку:



3. Пока мы не подключим сетку и навигатор к таблице, данные не будут отображаться. Сохраните проект в отдельную папку под именем **IBXTrans.**
4. Поскольку и навигатор, и сетка нам нужны только для чтения данных, выделите навигатор, откройте сложное свойство **VisibleButtons** и сделайте невидимыми все

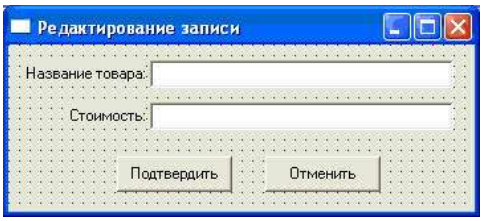
кнопки, кроме **nbFirst**, **nbPrior**, **nbNext**, **nbLast** и **nbRefresh**. Отрегулируйте ширину навигатора, чтобы кнопки стали квадратными, как на рисунке выше. У сетки свойство **ReadOnly** переведите в **True**.

5. Командой **File - New - Data Module** создайте новый модуль данных. Свойство **Name** окна переименуйте в **fDM**, а сам модуль сохраните под именем **DM**. Перейдите в главное окно, и командой **File - Use Unit** подключите к нему модуль **DM.**
6. В модуль поместите следующие компоненты из вкладки **InterBase**: **IBDatabase**, **IBTable**, **IBQuery**, два компонента **IBTransaction** и один **DataSource** из вкладки **Data Access**.
7. Займемся настройкой этих компонентов. Выделите компонент **IBDatabase**. Его свойство **Name** переименуйте в **IBDB**, чтобы было короче. Откройте свойство **Params** (откроется окно редактора параметров). Там укажите следующие параметры: **user_name=sysdba, password=masterkey, lc_ctype=win1251**

Обратите внимание, все параметры можно писать маленькими буквами, пробелы перед и после знака «=« недопустимы.

8. Далее, в свойство **DatabaseName** найдите и поместите файл нашей базы данных **First.gdb**. Чтобы при загрузке программы не запрашивался логин и пароль, свойство **LoginPrompt** переведите в **False**. После этого свойство **Connected** переведите в **True.** Если вы все сделали правильно, база откроется, ошибок не возникнет.
9. Выделите первый **IBTransaction**, его свойство **Name** переименуйте в **IBTrans1**. Этот компонент будет создавать транзакции для считывания данных в сетку и навигатор на главной форме. Откройте свойство **Params** (откроется окно редактора параметров), и впишите следующие параметры: **read, read_committed, rec_version**.
10. В свойстве **DefaultDatabase** выберите **IBDB**, а у **IBDB** в свойстве **DefaultTransaction** в свою очередь, выберите **IBTrans1**. После чего переведите свойство **Active** транзакции **IBTrans1** в **True**.
11. Вторую транзакцию назовите **IBTrans2**. Она будет нужна только для записи новых данных в таблицу. Поэтому в свойстве **Params** мы введем следующие параметры: **write, concurrency, nowait**.
12. В свойстве **DefaultDatabase** этой транзакции также выберите **IBDB**, а вот свойство **Active** оставьте **False** - транзакция будет автоматически запускаться, когда мы попытаемся изменить данные в таблице.

13. Перейдем к компоненту **IBTable**. Свойство **Name** переименуйте в **TTovar**, в свойстве **Database** выберите **IBDB**, в свойстве **Transaction** выберите читающую транзакцию **IBTrans1**. Теперь в свойстве **TableName** найдите и откройте таблицу **TOVAR**, после чего свойство **Active** переведите в **True**. Таблица открыта.
14. Выделите **DataSource** и переименуйте **Name** в **DSTovar**. В свойстве **DataSet** выберите таблицу **TTovar**. Теперь можно перейти на главную форму, выделить сетку и навигатор, и в их свойстве **DataSource** выбрать **fDM.DSTovar**. Данные должны отобразиться в сетке, а некоторые кнопки навигатора станут недоступны.
15. Вернемся в модуль данных. Выделите запрос **IBQuery**. Свойство **Name** переименуйте в **Q1**. В свойстве **Database** выберите **IBDB**, а в свойстве **Transaction** - **IBTrans2**. Более ничего делать не нужно, запросы будем строить программно.
16. Для добавления новых записей и редактирования существующих создадим еще одну форму командой **File - New - Form**. Окно редактора будет очень простым:



17. Форму назовите **fEditor**, сохраните в папку проекта, дав модулю имя **Editor** (так как последним мы открывали файл **First.gdb**, то при попытке сохранения окна **Delphi** по умолчанию предложит папку с БД. Измените ее на папку с проектом.). Командой **File - Use Unit** подключите к редактору модуль **DM**. На форме расположите два компонента **Label**, два простых **Edit** и две кнопки. Не помешает в свойстве **BorderStyle** формы выбрать **bsDialog**, а в свойстве **Position** - **poMainFormCenter**. И не забудьте очистить текст у компонентов **Edit**, а у кнопок и компонентов **Label** изменить свойство **Caption** в соответствии с рисунком.
18. Вернемся к главной форме. Командой **File - Use Unit** подключите к главной форме модуль **Editor**. Сгенерируйте событие нажатия на кнопку «**Добавить запись**». В полученной процедуре впишите код закрытия **Q1** (на случай, если ранее запрос был активен), и вызова редактора:

```
{Добавить запись}
procedure TfMain.Button1Click(Sender: TObject);
begin
```

```
 fDM.Q1.Close;
 fEditor.ShowModal;
end;
```

19. Таким образом, мы будем добавлять новую запись. А для редактирования существующей записи, выделите сетку **DBGrid** и сгенерируйте для нее событие **OnDblClick**. То есть, открывать редактор мы будем по двойному щелчку на записи. При этом в таблице **TTovar** станет текущей нужная нам запись. Прежде, чем вызывать редактор, нам нужно в **Q1** получить нужную запись. Для этого мы создадим запрос, откроем **Q1** и только потом вызовем редактор. Итак, код события **OnDblClick** следующий:

```
{Двойной щелчок по сетке - редактируем запись}
procedure TfMain.DBGrid1DblClick(Sender: TObject);
begin
 fDM.Q1.SQL.Clear;
 fDM.Q1.SQL.Add('select * from Tovar where ID = '+
   IntToStr(fDM.TTovar.FieldByName('ID').AsInteger));
 fDM.Q1.Open;
 fEditor.ShowModal;
end;
```

20. Как видно из кода, в запросе **Q1** мы генерируем запрос вроде такого:

```
SELECT *
FROM TOVAR
WHERE ID = 3
```

21. Разумеется, номер **ID** будет зависеть от того, по какой записи мы щелкнули. При открытии **Q1**, в нее попадет лишь одна интересующая нас запись. Больше ничего в главной форме делать не нужно. Переходим к окну редактора.
22. Тут у нас может быть два варианта: либо мы добавляем новую запись (**Q1** закрыта), либо редактируем существующую (**Q1**открыта и содержит нужную запись). В первом случае нам нужно будет очистить компоненты **Edit**, если там был текст, а во втором наоборот, вписать в них значения полей **Nazvanie** и **Stoimost**.
23. В раздел глобальных переменных добавим переменную **i**, необходимую для хранения **ID** записи:

```
var
  fEditor: TfEditor;
  i: Integer; //идетификатор записи
```

24. Затем выделим форму редактора, и сгенерируем для нее событие **OnShow**. Код события следующий:

```
{При показе редактора}
procedure TfEditor.FormShow(Sender: TObject);
begin
  if fDM.Q1.Active then
    i := fDM.Q1.Fields[0].AsInteger
  else i := 0;
  //очищаем или заполняем эдиты:
  if i = 0 then begin
    Edit1.Text := '';
    Edit2.Text := '';
  end //if
  else begin
    Edit1.Text := fDM.Q1.Fields[1].AsString;
    Edit2.Text := fDM.Q1.Fields[2].AsString;
  end; //esle
end;
```

25. Как видно из приведенного кода, как только форма станет видимой, мы проверяем - активна ли **Q1**. Если да, то в переменную **i** прописываем идентификатор текущей записи, иначе **i** делаем равным нулю. Это нам нужно для того, чтобы в дальнейшем знать, с какой записью работать. Ведь в **Q1** будут помещаться другие запросы, и она уже не будет содержать нужную запись.
26. Далее, если **i = 0** (новая запись), мы очищаем компоненты **Edit**, иначе считываем в них значения второго и третьего поля запроса (**Nazvanie** и **Stoimost**).
27. Пойдем дальше. Для кнопки «**Отменить**» введем код простого закрытия формы:

```
//закрываем форму:
  Close;
```

28. Так как пользователь может закрыть окно не только кнопкой «Отменить», но и

клавишами <Alt + F4> или просто нажав на крестик в правом верхнем углу окна, то проверку на активность транзакции нужно делать в событии формы OnClose.

29. Сгенерируем для формы редактора событие OnClose, в котором будем проверять, не в работе ли транзакция IBTrans2, и если да, то сделаем откат транзакции:

```
if fDM.IBTrans2.InTransaction then
   fDM.IBTrans2.RollbackRetaining;
```

30. Прежде, чем перейдем к кнопке «**Подтвердить**», подумаем вот о чем: для **InterBase** в вещественных числах между целой частью и дробной должен быть разделитель точка, а пользователь может ввести запятую. Кроме того, в существующих записях разделитель также отображается как запятая, и если мы при показе формы автоматически заполним **Edit2**, то там тоже будет запятая. Значит, для **Edit2** сгенерируем событие **OnChange**, где устроим простую проверку:

```
{Изменение данных в Edit2}
procedure TfEditor.Edit2Change(Sender: TObject);
var
  ind: Byte;
  s: String;
begin
  s:= Edit2.Text;
  for ind:= 1 to Length(s) do
    if s[ind] = ',' then s[ind]:= '.';
  Edit2.Text := s;
end;
```

Здесь, если в тексте будет обнаружена запятая, то она автоматически поменяется на точку. Однако приложение демонстрационное, поэтому мы не делаем проверку на то, что пользователь может ввести не только цифру, точку или запятую, но и какой-нибудь другой символ, например, пробел или букву. Вы можете сделать более детальную проверку, если желаете.

31. Нам осталось лишь сгенерировать нажатие на кнопку «**Подтвердить**». И здесь у нас будет два варианта: либо мы добавляем новую запись (**i = 0**), и тогда мы используем оператор **INSERT**, либо мы изменяем существующую. В последнем случае будет применяться оператор **UPDATE**. Код нажатия на кнопку следующий:

```
{Подтвердить}
procedure TfEditor.Button1Click(Sender: TObject);
begin
  //очищаем SQL-запрос:
  fDM.Q1.SQL.Clear;
  //создаем новый запрос, в зависимости от показателя i:
  if i = 0 then begin //если i = 0, то это новая запись
    fDM.Q1.SQL.Add('insert into Tovar(Nazvanie, Stoimost)');
    fDM.Q1.SQL.Add('values('+QuotedStr(Edit1.Text)+', '+Edit2.Text+')');
  end //if
  else begin //модифицируем существующую запись
    fDM.Q1.SQL.Add('update Tovar set Nazvanie='+
    QuotedStr(Edit1.Text)+
                ', Stoimost='+
     Edit2.Text+' where ID ='+IntToStr(i));
  end; //else
  try
    //производим изменения:
    fDM.Q1.ExecSQL;
    //подтверждаем транзакцию:
    fDM.IBTrans2.CommitRetaining;
  except
    ShowMessage('Изменения данных не прошли!');
    fDM.IBTrans2.RollbackRetaining;
  end; //try
  //обновляем НД TTovar:
  fDM.TTovar.Refresh;
  //закрываем форму:
  Close;
end;
```

32. В случае добавления записи формируется запрос типа:

INSERT INTO TOVAR(Nazvanie, Stoimost) VALUES('Товар', 10.00)

33. В случае редактирования существующей записи формируется другой запрос:

UPDATE TOVAR

SET Nazvanie = 'Товар', Stoimost = 10.00

WHERE ID = 3

34. Разумеется, название товара, его стоимость и идентификатор **ID** будут зависеть от того, что именно введет пользователь в компоненты **Edit**, и какую строку в таблице выберет.
35. Обратите внимание, что у компонента **IBTrans2** вместо методов **Commit** (подтверждение) и **Rollback** (откат) используются методы **CommitRetaining** и **RollbackRetaining**, которые также подтверждают или откатывают транзакцию, но не закрывают ее при этом, оставляя активной. В многопользовательской среде, где идет активная работа с базой данных, транзакции лучше закрывать, чтобы в базе данных не «висело» множество активных транзакций.
36. Сохраните проект, скомпилируйте и попробуйте вводить новые значения и редактировать существующие. Проект использует две транзакции: для чтения с «мягкими» параметрами, и для записи с более «жесткими».

В проектах, которые интенсивно работают с базой данных, совершенно недопустимым будет использование только одной транзакции, это может привести к многочисленным конфликтам. Также недопустимым будет использование отдельной транзакции для каждого набора данных. Находите компромисс: разделяйте наборы данных на читающие, пишущие и НД для отчетов. И для каждой группы используйте отдельный компонент **IBTransaction**, с соответствующими задаче параметрами.

Внеаудиторная самостоятельная работа:

Составить письменно в тетради опорный конспект по основным компонентам по работе с транзакциями в **InterBase**.

Практическая работа №24

Формирование и вывод отчетов. Назначение и виды отчетов. Компоненты для формирования отчетов.

Цель работы: сформировать умения по формированию и выводу отчетов в **InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и

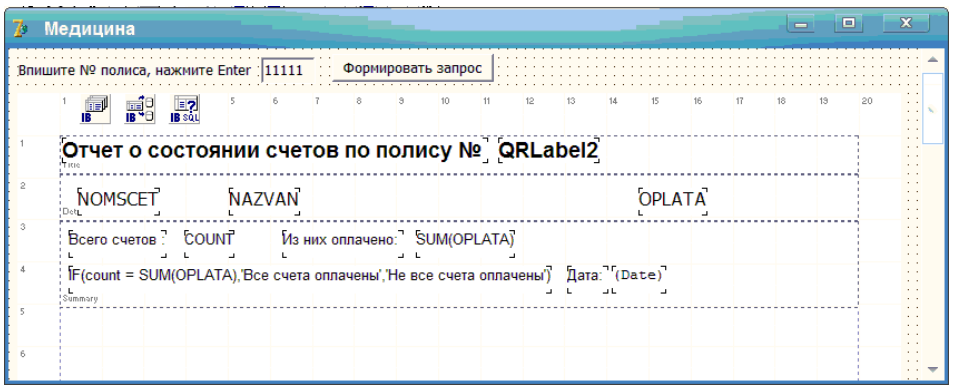
качество.

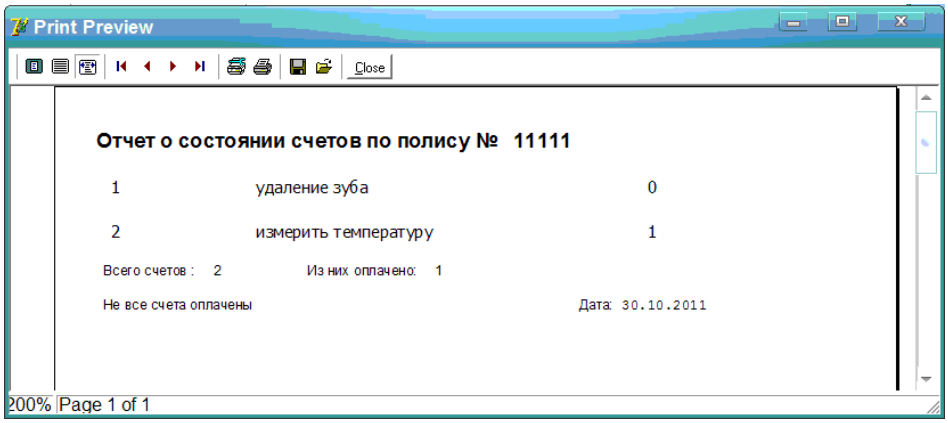
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**, интегрированная среда разработчика **Delphi**.

Задание:

1. Создайте базу данных формата **InterBase** для ведения компьютеризированного учета медицинских записей.
2. Разработайте приложение в **Delphi**, позволяющее сформировать и распечатать отчет о текущем состоянии счетов за оказанные медуслуги для данного клиента.





Методические указания по выполнению задания:

1. Создайте базу данных **C:\dbase1\MED.GDB** .
2. Создайте таблицы, используя **IBConsole** и SQL-запросы:

CREATE TABLE "CLIENT"
("NOMPOLIS" SMALLINT NOT NULL,
"FIO" CHAR(30) CHARACTER SET WIN1251 NOT NULL COLLATE PXW_CYRL,
"ADRES" CHAR(30) CHARACTER SET WIN1251 NOT NULL COLLATE PXW_CYRL,
"PASP" CHAR(15) CHARACTER SET WIN1251 NOT NULL COLLATE PXW_CYRL,
"DATANAC" DATE NOT NULL,
"DATAKON" DATE NOT NULL,
PRIMARY KEY ("NOMPOLIS"))

CREATE TABLE "SCETA"
("NOMSCET" SMALLINT NOT NULL,
"NOMPOLIS" SMALLINT NOT NULL,
"KODUSL" SMALLINT NOT NULL,
"OPLATA" SMALLINT NOT NULL,
PRIMARY KEY ("NOMSCET"))

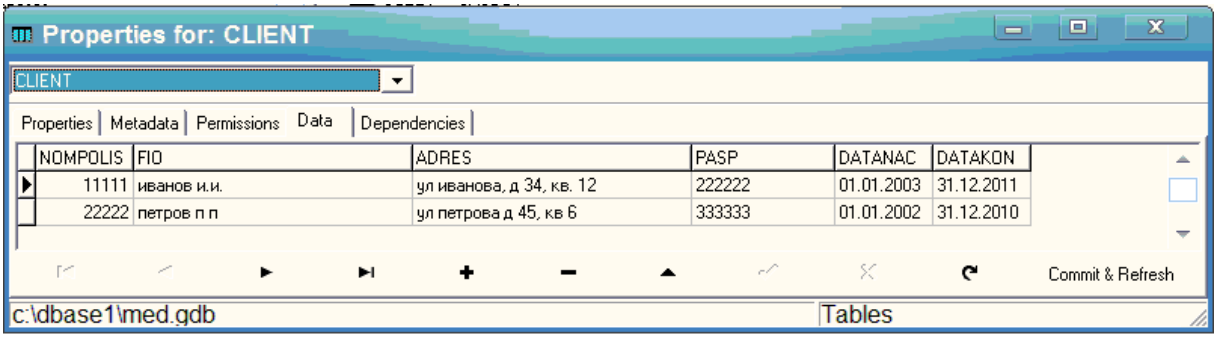
CREATE TABLE "USLUGI"
("KODUSL" SMALLINT NOT NULL,

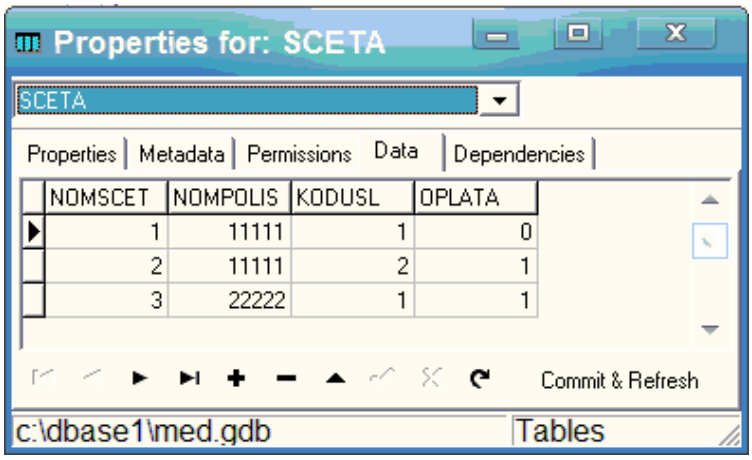
"NAZVAN" CHAR(20) CHARACTER SET WIN1251 NOT NULL COLLATE PXW_CYRL,

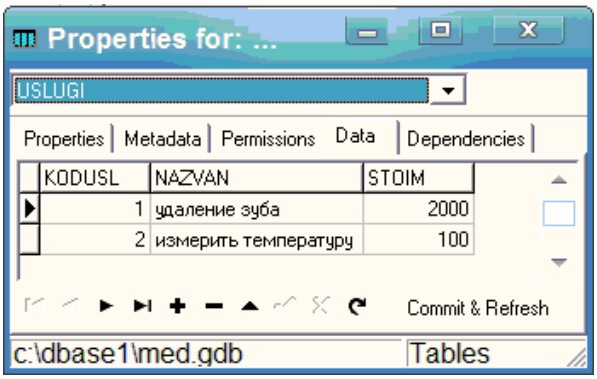
"STOIM" FLOAT NOT NULL,

PRIMARY KEY ("KODUSL"))

3. Заполните таблицы данными любым из известных вами способов.







4. **QuickReport** идет в поставке **Delphi**, но по умолчанию его нет в палитре компонентов. Для добавления **QuickReport** в палитру компонент нужно выполнить следующую последовательность действий:
   - В главном меню **Delphi** выбрать пункт меню **Component - Install Packages...**

- Откроется окно управления загружаемыми пакетами. В нем нужно нажать кнопку **Add...** Для добавления нового пакета.
- Откроется окно выбора файлов. В данном окне нужно перейти в папку куда была установлена **Delphi** (за частую **это C:\Program Files\Borland\Delphi**\) и открыть папку **Bin**.
- В окне выбора файлов установим Тип файлов в **Package library.bpl**
- Выберем пакет с именем **dclqrt70.bpl** и нажмем кнопку **Открыть** для подтверждения выбора, после чего в списке установленных пакетов добавится **QuickReport Components**
- Нажимаем ОК для закрытия окно.

5. Создаем новый проект и тестируем работоспособность. Запускаем **Delphi.**
6. Устанавливаем на форму следующие компоненты и задаем их свойства:
   - **IBDatabase1, IBTransaction1, IBQuery1**,
   - С закладки **QReport: QuickRep1, QRBand1** (полоса заголовка отчёта). .**BandType= rbTitle**,
   - **QRBand2** (полоса итогов), **QRBand3** (полоса данных (поля результата запроса)).
   - На **QRBand1** устанавливаем: **QRLabel1** (Отчет о состоянии счетов по полису №),
   - **QRLabel2** (здесь будет № полиса)
   - На **QRBand3 (.BandType=rbDetail**) устанавливаем: **QRDBText1, QRDBText2, QRDBText3.** Выполним для них настройку: **.DataSet = IBQuery1, .DataField = NOMSCET, NAZVAN, OPLATA** (соответственно).
   - На **QRBand2** (.**BandType= rbSummary**) устанавливаем: **QRLabel3, QRLabel4, QRLabel5**, для вывода выражений: **QRExpr1, QRExpr2, QRExpr3**.
7. Настраиваем компоненты: **IBDatabase1: Database=C:\dbase1\MED.GDB**, на панели **Database Parameters** вписываем: **User Name: SYSDBA, Password : masterkey, Character set: WIN1251**, убираем флажок **Login prompt**, щелкаем **ОК**. Согласитесь на отключение БД, если будет запрос.
8. Далее: **IBDatabase1.DefaultTransaction = IBTransaction1, IBTransaction1.Default = IBDatabase1, IBDatabase1.Connected = true; IBTransaction1.Active = true; IBQuery1.Database = IBDatabase1; IBQuery1.SQL = Select NOMSCET, OPLATA, NAZVAN from Sceta, Uslugi where (Sceta.KodUsl = Uslugi.KodUsl)**,

QuickRep1.DataSet = IBQuery1; QuickRep1.Units = MM.

9. **Текст модуля unMed1**

```
unit unMed1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QuickRpt, QRCtrls, StdCtrls, ExtCtrls, DB, IBCustomDataSet,
  IBQuery, IBDatabase;

type
  TfrmMed = class(TForm)
    QuickRep1: TQuickRep;
    Label1: TLabel;
    edNom: TEdit;
    btnSQL: TButton;
    IBDatabase1: TIBDatabase;
    IBTransaction1: TIBTransaction;
    IBQuery1: TIBQuery;
    QRBand1: TQRBand;
    QRBand2: TQRBand;
    QRBand3: TQRBand;
    QRLabel1: TQRLabel;
    QRLabel2: TQRLabel;
    QRDBText1: TQRDBText;
    QRDBText2: TQRDBText;
    QRDBText3: TQRDBText;
    QRSysData1: TQRSysData;
    QRLabel3: TQRLabel;
    QRExpr1: TQRExpr;
    QRExpr2: TQRExpr;
```

```
    QRExpr3: TQRExpr;
    QRLabel4: TQRLabel;
    QRLabel5: TQRLabel;
    procedure edNomKeyPress(Sender: TObject; var Key: Char);
    procedure btnSQLClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmMed: TfrmMed;

implementation

{$R *.dfm}

procedure TfrmMed.edNomKeyPress(Sender: TObject; var Key: Char);
begin
  if key = #13 then
    QRLabel2.Caption:=edNom.Text;
end;

procedure TfrmMed.btnSQLClick(Sender: TObject);
begin
QRLabel2.Caption:= edNom.Text;
IBQuery1.Active:=false;
IBQuery1.SQL.Clear;
IBQuery1.SQL.append('Select NOMSCET, Oplata, NAZVAN from Sceta, Uslugi ');
IBQuery1.SQL.append('where (Sceta.KodUsl = Uslugi.KodUsl) ');
IBQuery1.SQL.append('and (Sceta.NOMPOLIS =''' + edNom.Text+''')');
```

```
IBQuery1.Active:=true;
QuickRep1.Preview;

end;
end.
```

Внеаудиторная самостоятельная работа:

Составить опорный конспект письменно в тетради по основным командам формирования отчетов.

Практическая работа №25

Установка привилегий доступа к данным.

Программное администрирование баз данных InterBase

Цель работы: сформировать умения по установке привилегий доступа к данным, научиться выполнять программное администрирование баз данных **InterBase**.

Реализуемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.
- ПК 2.5. Оформлять программную документацию в соответствии с принятыми стандартами.

Оборудование, технические и программные средства: персональный компьютер, сервер баз данных **Borland InterBase**, интегрированная среда разработчика **Delphi**.

Задание 1. Реализация резервного копирования

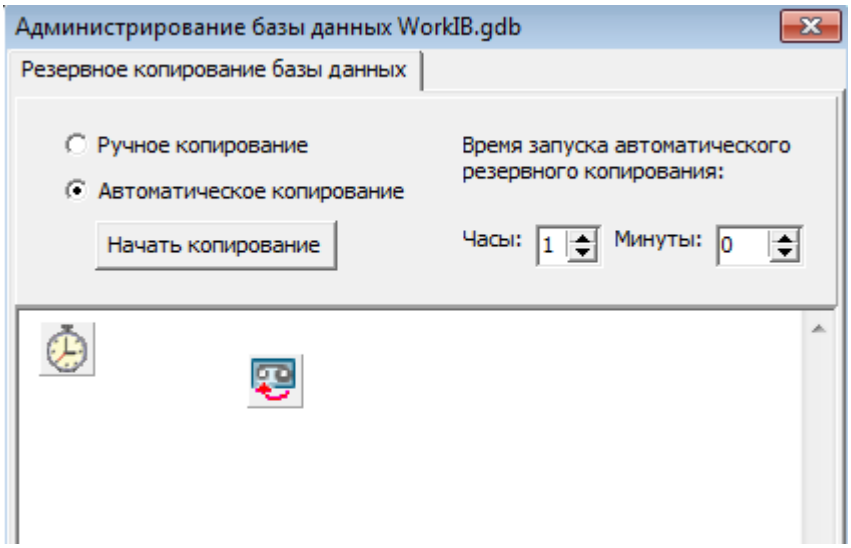
Требуется создать приложение, которое позволит регулярно делать резервную копию базы **WorkIB.gdb** как в ручном, так и в автоматическом режиме. Также приложение, в случае необходимости, должно уметь восстанавливать базу данных из резервной копии, и сохранять лог-файлы о копировании и восстановлении. В случае каких-то проблем, администратор вышлет вам эти логи, по которым вы сможете определить проблему. Также программа должна уметь добавлять новых пользователей, удалять или редактировать старых. Разумеется, пользователя **SYSDBA** удалять нельзя.

Методические указания по выполнению задания:

1. Убедитесь, что сервер **InterBase** работает, и загрузите **Delphi**.
2. Выделите форму, в свойство **Name** которой впишите **fMain**, а в свойство **Caption** – «**Администрирование базы данных WorkIB.gdb**».
3. Сохраните проект в отдельную папку, модулю дайте имя **Main**, а проекту в целом - **AdminIB**. Неплохо было бы сразу установить свойство **BorderStyle** в **bsDialog**, чтобы программа не могла менять размеры, а свойство **Position** - в **poDesktopCenter.**
4. Перейдите на вкладку **Win32 Палитры компонентов**, найдите и установите на форму компонент **PageControl** (каждая задача будет на своей вкладке). Переименуйте полученный **PageControl1** в **PC1** для краткости кода, а свойству **Align** присвойте значение **alClient**.
5. Щелкните по компоненту **PC1** правой кнопкой и выберите команду **New Page** - создалась новая вкладка. Выделите ее, и в свойстве **Name** вместо **TabSheet1** напишите **TSh1**, а в свойстве **Caption** – «Резервное копирование базы данных». Будьте внимательны: очень легко ошибиться, и вместо листа **TabSheet** выделить весь компонент **PageControl**. Примерный вид вкладки смотрите на рисунке:

6. Вначале установите на лист панель **Panel1**, свойству **Align** которой присвойте значение **alTop**. Очистите свойство **Caption**. На панель поместите две радиокнопки **RadioButton**. В свойстве **Name** первой кнопки укажите **RB1**, у второй - **RB2**. А в свойстве **Caption**, соответственно, **Ручное копирование** и **Автоматическое копирование**. Свойство **Checked** компонента **RB2** переведем в **True.**
7. Ниже поместите простую кнопку **Button**, которую переименуйте в **bStartBackup**, а в свойстве **Caption** напишите **Начать копирование.** Свойство **Enabled** переведите в **False**. Поскольку по умолчанию, программа будет выполнять автоматическое копирование, эта кнопка будет недоступна. Как только мы выделим радиокнопку **Ручное копирование**, сразу вернем доступность этой кнопке.
8. В правой части панели поместите компонент **Label**. Чтобы компонент мог содержать многострочный текст, свойство **AutoSize** переведите в **False**, а свойство **WordWrap** - в True. В свойстве **Caption** напишите текст **Время запуска автоматического резервного копирования:**, и подгоните размеры, как на рисунке.
9. Ниже находятся еще два **Label** с текстом **Часы:** и **Минуты:**. Рядом с ними располагаются два компонента **SpinEdit** с вкладки **Samples** палитры компонентов. Первый переименуйте в **SE1**, второй - в **SE2**. У первого в параметре **MinValue** оставьте 0, в параметре **MaxValue** укажите **23**, а в параметре **Value** поставьте 1 (по умолчанию, автокопирование будет начинаться в час ночи, когда сотрудников на работе нет). У второго **SpinEdit** параметры такие: **MinValue = 0**, **MaxValue = 59, Value = 0**.
10. Под панелью установите компонент **Memo**. Свойство **Align** установите в **alClient**, свойство **ScrollBars** в **ssVertical**, и не забудьте очистить от текста свойство **Lines**.

11. Далее главное: нужно добавить два не визуальных компонента - **Timer** с вкладки **System** и **IBBackupService** с вкладки **InterBase Admin**. Компоненты не визуальные, их можно расположить на любом месте, первый будет запускать копирование автоматически, когда подойдет указанное время, второй - осуществлять само копирование. У **Timer** свойство **Interval** установите **60000**, чтобы таймер срабатывал один раз в минуту.
12. Компонент **IBBackupService** предназначен для создания резервных копий базы данных **InterBase**. Этот компонент позволяет делать различные настройки резервного копирования, в зависимости от того, какие параметры в свойстве **Options** включены. У **IBBackupService** свойство **Name** для краткости переименуйте в **IBBS**. На этом приготовления вкладки закончены, приступим к кодированию.



13. Все резервные копии должны попадать в одну папку, например, **D:\WorkIB\Backup**. Эта папка физически должна существовать на диске. При этом мы не сможем делать копии с одним и тем же именем, ведь операционная система не позволит создавать файл с таким же именем. Конечно, можно перед очередным копированием удалять старую копию, но это тоже не выход - что, если новая резервная копия будет создана с ошибками, то есть, в базе данных появились нарушения? Можно было бы восстановить ее из старой копии, но для этого ее не нужно удалять! Выход: добавлять к имени файла дату и время его создания, тогда можно делать сколько угодно копий, и у всех будут разные имена. Для этого в разделе **Private** модуля главной формы опишем функцию **GetName**:

```
private
  { Private declarations }
   function GetName():String;
```

14. Установите курсор на название функции и нажмите **Ctrl+Shift+C**, чтобы сгенерировать саму функцию. Вот ее код:

```
var
  ye, mo, da : Word; //для даты
  ho, mi, se, ms : Word; //для времени
  st : String[2]; //для добавления нуля, например 05
begin
  DecodeDate(Date, ye, mo, da); //декодируем на составные дату
  DecodeTime(Time, ho, mi, se, ms); //декодируем время
  //теперь собираем строку:
  Result:= IntToStr(ye); //добавили год
  //получаем и добавляем месяц:
  st:= IntToStr(mo);
  //если 1 символ, добавим спереди 0
  if Length(st) = 1 then st:= '0' + st;
    Result := Result + st;
  //получаем и добавляем день:
  st:= IntToStr(da);
  if Length(st) = 1 then st:= '0' + st;
  Result := Result + st + '_';
  //теперь получаем и добавляем час:
  st:= IntToStr(ho);
  if Length(st) = 1 then st:= '0' + st;
  Result := Result + st;
  //получаем и добавляем минуты:
  st:= IntToStr(mi);
  if Length(st) = 1 then st:= '0' + st;
  Result := Result + st;
  //получаем и добавляем секунды:
  st:= IntToStr(se);
  if Length(st) = 1 then st:= '0' + st;
  Result := Result + st + '_';
  //теперь функция вернет префикс имени файла, например:
  //20100205_010012_
end;
```

Здесь мы декодировали на составные части дату и время. Год уже имеет 4 цифры, поэтому его дополнительно обрабатывать не нужно. А вот месяц, день, час, минута или секунда могут состоять из одной цифры.

В переменной **st** мы получаем номер месяца. Если этот номер состоит из одной цифры, перед ней добавим **'0'**. И в конце прибавим результат в переменную **Result**. Таким же образом мы проверяем и остальные данные. Эта функция гарантирует нам не только

уникальность имени файла, но и правильную сортировку файлов по дате. К тому же по имени сразу видно - когда был создан файл.

15. Теперь сделаем доступной кнопку **bStartBackup**, если пользователь отметил радиокнопку **RB1** (Ручное копирование). Для этого сгенерируем для **RB1** событие **onClick**:

```
{Щелкнули по "Ручное копирование"}
procedure TfMain.RB1Click(Sender: TObject);
begin
   //если отмечено Ручное копирование, делаем
  //кнопку доступной:
  if RB1.Checked then
    bStartBackup.Enabled:= True;
end;
```

16. Точно также сделаем кнопку недоступной, если щелкнули по **RB2** (Автоматическое копирование). Для этого сгенерируем событие **onClick** для **RB2**:

```
{Щелкнули по "Автоматическое копирование"}
procedure TfMain.RB2Click(Sender: TObject);
begin
   //если отмечено Автоматическое копирование, делаем
  //кнопку недоступной:
  if RB2.Checked then
    bStartBackup.Enabled:= False;
end;
```

17. Копирование у нас будет начинаться либо по нажатию кнопки, либо по срабатыванию таймера, в зависимости от того, какая радиокнопка выделена. Значит, чтобы дважды не писать один и тот же код, создадим процедуру резервного копирования, которую будем вызывать из двух мест. Поскольку процедура будет напрямую работать с компонентом **IBBS**, вначале объявим ее в разделе **private**, после функции **GetName**:

```
private
  { Private declarations }
   function GetName():String;
   procedure BackupCopy;
```

18. Установите курсор на процедуру и нажмите **Ctrl+Shift+C**, чтобы сгенерировать тело процедуры. Ее код:

```
procedure TfMain.BackupCopy;
var s : String; //для получения префикса файла
begin
  //получим префикс:
  s:= GetName();
  //очистим Memo1, если там что то есть:
  Memo1.Clear;
  //задаем параметры компонента IBBackupService:
  IBBS.ServerName := 'localhost';
  IBBS.LoginPrompt:= False;
  IBBS.Params.Add('user_name=sysdba');
  IBBS.Params.Add('password=masterkey');
  IBBS.Active := True;
  //начинаем копирование
  try
    IBBS.Verbose:= True;
    IBBS.DatabaseName:= 'd:\Drozdova Ann\BaseIB\WorkIB\WorkIB.gdb';
    IBBS.BackupFile.Clear;
    IBBS.BackupFile.Add('d:\Drozdova Ann\BaseIB\WorkIB\Backup\' +
s + 'WorkIB.gbk');
    IBBS.ServiceStart;
    //пока не дошли до конца, записываем параллельно лог в Memo1:
    while not IBBS.Eof do
      Memo1.Lines.Add(IBBS.GetNextLine);
  finally
    IBBS.Active:= False;
  end;
  //сохраним лог в файл:
  Memo1.Lines.SaveToFile('d:\Drozdova Ann\BaseIB\WorkIB\Backup\' +
s + 'WorkIB.log');
end;
```

Здесь мы вначале получили в переменную префикс имени файла. Зачем это нужно? Нам требуется: сделать резервную копию, сохранить лог копирования в файл. Эти действия будут производиться в разное время, поэтому чтобы префикс копии совпадал с префиксом лог-файла, мы и сохраняем его в переменную **s**. Далее мы очищаем **Memo1**, если вдруг там уже был текст. После чего приступаем к настройкам параметров компонента **IBBS**.

Поскольку копирование будет производиться на серверном ПК, то будет использован локальный адрес, следовательно, свойство **ServerName** мы можем назвать, как угодно. Если бы сервер был сетевым, пришлось бы писать сетевое имя этого ПК, или его IP - адрес. При этом следует обратить внимание на свойство **Protocol**. Это свойство позволяет выбрать протокол соединения. Поскольку программа будет загружаться там же, где установлен **InterBase** и где хранится база данных, оставляет значение по

умолчанию - **Local**. Если программа будет работать с удаленным сервером, то здесь нужно будет выбрать один из сетевых протоколов, обычно выбирают **TCP.**

Далее с помощью **LoginPrompt** мы заявляем, что не нужно запрашивать имя пользователя и пароль (кто же введет эти данные в час ночи, когда запустится авто-копирование?). Затем мы вводим в свойство **Params** имя пользователя **SYSDBA** и его пароль. Если вы изменили пароль (а это обязательно нужно сделать в реальной системе), вместо **masterkey** укажите свой пароль. В конце мы делаем **IBBS** активным.

Само копирование мы помещаем в блок **try…finally…end**, поскольку копирование может быть и неудачным, если БД разрушена.

Здесь свойство **Verbose** (многословность) определяет - будет ли выводиться лог резервного копирования. При значении **True** лог ведется, иначе - нет. Нам нужно, чтобы лог создавался.

В свойство **DatabaseName** помещаем адрес и имя нашей рабочей базы данных.

Свойство **BackupFile** должно содержать имя файла (или файлов) резервной копии. У нас база данных состоит из одного файла, и резервная копия также будет состоять из одного файла, поэтому прежде, чем мы поместим туда имя очередной резервной копии, это свойство нужно очистить. Затем формируем имя файла из адреса, префикса и самого имени **WorkIB.gbk**. А затем стартуем копирование методом **ServiceStart**. Заметим, что установка **True** в свойстве **Active** компонента не начинает резервное копирование, а только делает компонент активным.

Параллельно копированию в компонент **Memo1** помещаем отчет о текущем действии, от начала копирования до конца. Метод **GetNextLine** компонента **IBBackupService** возвращает очередную строку лог-отчета. Когда копирование окончено, делаем компонент **IBBS** неактивным.

И в самом конце процедуры записываем полученный в **Memo1** лог в файл с таким же именем, как резервная копия, но расширением ***.log**

19. Теперь для ручного копирования нам осталось сгенерировать процедуру нажатия на кнопку **Начать копирование**, откуда вызовем процедуру **BackupCopy**:

```
  {Нажали на кнопку "Начать копирование"}
procedure TfMain.bStartBackupClick(Sender: TObject);
begin
  BackupCopy;
end;
```

20. Попробуйте выполнить ручное копирование. Напомним, что сервер **InterBase** должен быть запущен, а папка **D:\WorkIB\Backup** должна физически существовать на диске. Как только вы нажмете на кнопку, копирование начнется. Закончится оно примерно такой строкой: **gbak: closing file, committing and finishing. 38912 bytes written**. Зайдите в папку **Backup** - там должна появиться пара файлов с одинаковым именем и расширениями ***.gbk** и ***.log**. Это и есть наши резервная копия и лог-файл.
21. Займемся автоматическим копированием. Выделите таймер и сгенерируйте для него событие **onTimer**, которое будет срабатывать каждую минуту (Interval =60000):

```
procedure TfMain.Timer1Timer(Sender: TObject);
var
  ho, mi, se, ms : Word; //для времени
begin
  //если включено ручное копирование, просто выходим:
  if RB1.Checked then Exit;
  //иначе декодируем время
  DecodeTime(Time, ho, mi, se, ms); //декодируем время
  //если нужный час и нужная минута, начинаем копирование:
  if (ho = SE1.Value) and (mi = SE2.Value) then BackupCopy;
end;
```

Вначале мы проверяем, не включено ли ручное копирование. Если включено, то сразу выходим, ничего не делая. Затем мы снова декодируем время, чтобы посмотреть - совпадают ли данные в компонентах **SpinEdit** с текущими часом и минутой. И если совпадают, вызываем процедуру **BackupCopy**.

Заметим, что таймер может сработать не сразу, как системные часы покажут нужное время. Ведь в таймере учитываются и секунды. Значит, если копирование у нас должно начаться в 01:00, а когда программа была запущена, ваше системное время показывало, скажем, 25 секунд, то и таймер сработает в 01:00:25. Что, в принципе, не столь важно.

22. Сохраните проект, скомпилируйте и запустите. Попробуйте установить время авто-копирования на минуту-другую больше, чем показывают ваши системные часы (должна быть выделена радиокнопка **Автоматическое копирование**). Дождитесь, когда системное время сравняется с указанным. Возможно, придется подождать еще несколько секунд, после чего резервное копирование начнется автоматически. Опять появится пара файлов в папке **Backup**. Таким образом, достаточно выделить радиокнопку **RB2** (у нас она выделена по умолчанию), и не выключать на сервере программу, чтобы обеспечить регулярное резервное копирование базы данных. Разумеется, серверный ПК с **InterBase**, вашей базой данных и администраторской

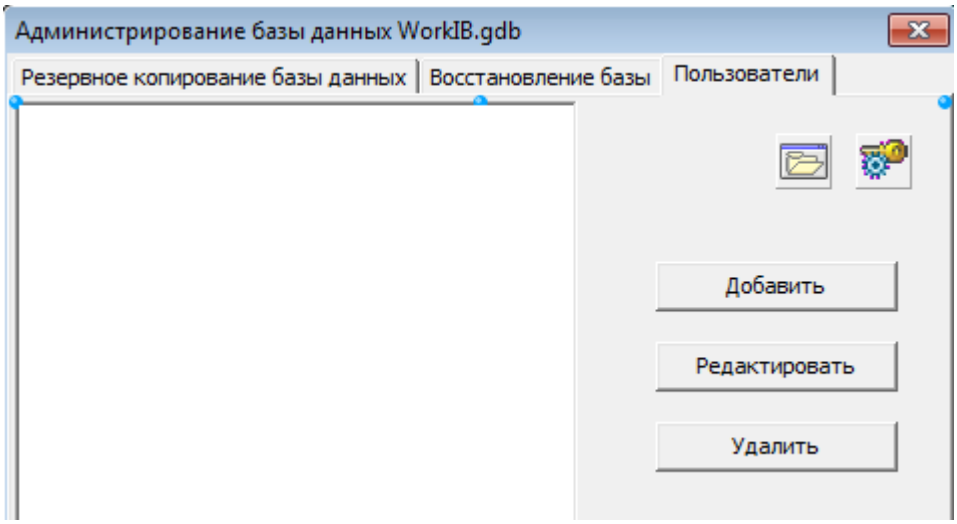
программой должен быть включен в назначенное время (серверные компьютеры обычно работают в режиме 24/7, то есть 24 часа в сутки, 7 дней в неделю).

Задание 2. Работа с пользователями

Нам нужно, чтобы приложение выполняло: добавление нового пользователя, изменение данных существующего пользователя (пароль, имя, отчество, фамилия). Если пользователь - **SYSDBA**, редактирование запрещаем. Ведь при резервном копировании и восстановлении мы жестко задавали имя **SYSDBA** и пароль **masterkey** (если вы изменили этот пароль, значит у вас свой вариант). И если администратор изменит этот пароль, программа не сможет работать, вам придется изменить этот пароль в программе и перекомпилировать ее. Впрочем, в дальнейшем вы можете усложнить программу, добавив в нее, например, работу с ini -файлом. В этом случае, вы сможете прописать в ini -файл новый пароль пользователя **SYSDBA**, если администратор изменит его, и в дальнейшем использовать новый пароль. Если же администратор не будет менять пароля, то можно использовать пароль по умолчанию, «**masterkey**» (или ваш вариант). Удаление любого пользователя, кроме **SYSDBA**.

Методические указания по выполнению задания:

1. Для реализации этой задачи создадим новую вкладку, которую назовем **TSh3**, а в свойстве **Caption** напишем «**Пользователи**». Внешний вид вкладки показан на следующем рисунке:
2. Здесь мы установили компонент **ListBox**, свойство **Name** которого переименовали в **LB1**, а свойство **Align** перевели в **alLeft**, чтобы компонент занял всю левую часть окна. Измените размеры компонента, как на рисунке.



3. В правой части установили три простых кнопки **Button**. Свойства **Name** кнопок переименовали в **bAddUser, bModifyUser** и **bDeleteUser**, а в свойстве **Caption**, соответственно, прописали «**Добавить**», «**Редактировать**» и «**Удалить**».
4. Кроме того, мы добавили не визуальный компонент **IBSecurityService** с вкладки **InterBase Admin**. Компонент для краткости обращения переименовали в **IBSS**. Этот компонент предназначен для работы с пользователями, зарегистрированными в **InterBase**, и позволит нам редактировать, добавлять и удалять пользователей.
5. Несмотря на кажущуюся простоту окна, реализация работы с пользователями немного сложнее предыдущих примеров. В отличие от компонентов **IBBackupService** и **IBRestoreService**, компонент **IBSecurityService** выполняет не одну, а три задачи, что выражено кнопками в правой части окна. Кроме того, **IBSecurityService** работает не с нашей базой данных **WorkIB.gdb**, а с системной БД **InterBase isc4.gdb**, что накладывает некоторые ограничения. Например, имя, отчество и фамилию пользователя не получится вносить русскими буквами, так как **isc4.gdb** создавалась не в кодировке **WIN1251**. Кроме того, в целях безопасности, компонент **IBSecurityService** не выводит существующий пароль пользователя, хотя и позволяет менять его. Компонент позволяет оперировать такими данными, как логин пользователя, его пароль, имя, отчество, фамилия, идентификатор пользователя **UserID** и идентификатор группы **GroupID.**

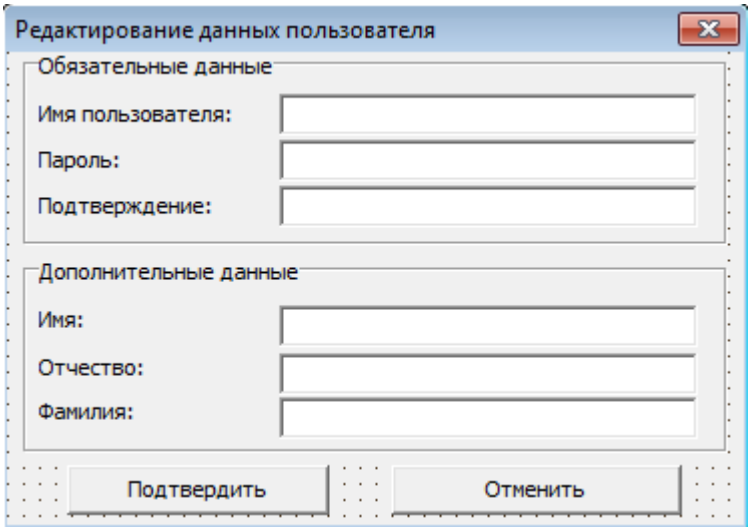
   Для доступа к данным пользователя компонент **IBSecurityService** имеет свойство **UserInfo**, которое представляет собой список пользователей. Индекс свойства начинается с 0, как и список строк **ListBox**. Так как индексы **ListBox.Items** и **IBSecurityService**. **UserInfo** будут совпадать, это сильно облегчит нашу задачу.
6. Первым делом нам нужно при создании главной формы заполнить компонент **IBSecurityService** свежими данными о пользователях, и обновить этими данными список **ListBox**. Так как нам то же самое придется делать при добавлении, редактировании и удалении пользователей, в разделе **private** добавим новую процедуру **ReloadUsers**. Её реализация представлена ниже:

```
procedure TfMain.ReloadUsers;
var i: Integer; //для счетчика
begin
  //вначале очистим ListBox:
  LB1.Clear;
  //заполняем список пользователями:
  IBSS.ServerName:= 'localhost';
  IBSS.LoginPrompt:= False;
  IBSS.Params.Add('user_name=sysdba');
  IBSS.Params.Add('password=masterkey');
  IBSS.Active:= True;
  try
    IBSS.DisplayUsers; //получаем  информацию о пользователях
    for i:=0 to IBSS.UserInfoCount -1 do
      LB1.Items.Add(IBSS.UserInfo[i].UserName);
  finally
    IBSS.Active:= False;
  end; //try
end;
```

Здесь все достаточно прозрачно. Вначале очищаем **ListBox** от возможных старых записей. Затем настраиваем сервис **IBSecurityService**, так же, как ранее настраивали компоненты **IBRestoreService** и **IBBackupService**. Метод **DisplayUsers** получает в компонент всю информацию о пользователях, после чего она доступна в свойстве **UserInfo**. Свойство **UserInfoCount** показывает общее количество пользователей. Подсвойство **UserName** свойства **UserInfo** содержит логин текущего пользователя. Мы добавляем в ListBox имя очередного пользователя.

7. Теперь нужно вызвать эту процедуру при создании формы. Для этого сгенерируйте для главной формы событие **OnCreate**, в котором сделаем вызов этой процедуры: **ReloadUsers**.
8. Проверить результат можно, скомпилировав и загрузив программу. При загрузке, в окне **ListBox** должны отобразиться зарегистрированные пользователи.
9. Далее нам придется сделать новую форму - редактор данных пользователей. Выберите команду **File - New - Form**. Форму назовите **fEditor**, в свойстве **Caption** напишите «**Редактирование данных пользователя**», сохраните новый модуль под именем **Editor**.
10. Свойство **BorderStyle** формы переведем в **bsDialog**, чтобы пользователь не мог менять размеры окна, а в свойстве **Position** выберем **poMainFormCenter**.
11. Внешний вид новой формы показан на рисунке ниже:

12. Итак, у нас имеется: два компонента **GroupBox**, шесть компонентов **Label**, шесть компонентов **Edit**, две кнопки **Button**. В свойстве **Caption** первого **GroupBox** напишем «**Обязательные данные**», а второго - «**Дополнительные данные**».
13. На верхний **GroupBox** помещаем три **Label** и заполняем текст, как на рисунке. Затем помещаем три **Edit**. У первого свойство **Name** переименуем в **eUser**, у второго - **ePass**, и у третьего - **ePass2**.
14. Выделите одновременно (с нажатой **Shift**) компоненты **ePass** и **ePass2**. В свойстве **PasswordChar** укажите символ «*», чтобы скрыть реальный пароль. Не забудьте удалить текст из свойства **Text** всех компонентов **Edit**.
15. На нижний **GroupBox** также помещаем три **Label** и заполняем текст, как на рисунке. Затем помещаем три **Edit**. У первого свойство **Name** переименуем в **eName**, у второго - **eMiddle**, и у третьего - **eLast**. Очищаем свойство **Text** у всех **Edit**.
16. Ниже расположим две кнопки, которые переименуем соответственно, в **bAccept** и **bCancel**, а свойства **Caption** - как на рисунке.



17. Данная форма будет показываться в двух случаях: при добавлении нового пользователя, и при редактировании существующего. В первом случае пользователь еще не имеет пароля, во втором - имеет. Причем при редактировании существующего пользователя, администратор может как сменить старый пароль, так и не трогать его. Как узнать, менялся ли пароль существующего пользователя, если мы не имеем возможности вывести его с помощью **IBSecurityService**? Вариант: при редактировании существующего пользователя поместим в **ePass** и **ePass2** какой-нибудь длинный пароль «по умолчанию», например, двадцать единичек. А при сохранении результата

редактирования будем смотреть - если в **ePass** пароль «по умолчанию», значит сохранять пароль не нужно. В случае же добавления нового пользователя **ePass** и **ePass2** заполняться не будут.

18. Пойдем дальше. Как уже говорилось, база данных **isc4.gdb** не поддерживает кодировку **win1251**, следовательно, русские буквы в нее помещать нельзя. Значит, при вводе текста во все шесть компонентов **Edit** придется делать проверку - что вводит пользователь. Если английские буквы или цифры, или **BackSpace**, то ничего не делаем, иначе запрещаем символ. Чтобы шесть раз не писать один и тот же код, в разделе **private** создадим функцию **KeyCan**:

```
private
  { Private declarations }
   function KeyCan(c:Char):Boolean;
```

19. Код реализации функции представлен ниже:

```
function TfMain.KeyCan(c: Char): Boolean;
begin
case c of
    'A'..'z': Result:= True; //англ. буквы разрешаем
    '0'..'9': Result:= True; //цифры разрешаем
    #8 : Result:= True; //BackSpace разрешаем
  else Result:= False; //остальное запрещаем
  end; //case
end;
```

20. Теперь нам нужно реализовать проверку во всех шести **Edit**. Выделите первый, и сгенерируйте для него событие **OnKeyPress**. В процедуру поместите только одну строку: **if not KeyCan(Key) then Key:= #0;**
21. То же самое проделайте с оставшимися пятью **Edit**. Таким образом, мы реализовали проверку на ввод допустимых символов. Если пользователь попытается ввести что-нибудь, кроме английских букв, цифр или **BackSpace**, то его действия будут игнорироваться.
22. Пойдем дальше. Если мы редактируем нового пользователя, то в **ePass** и **ePass2** у нас будет текст в двадцать единиц. Если администратор захочет изменить пароль, то он изменит текст в **ePass**. Значит, для **ePass** нужно сгенерировать событие **OnChange**, в котором очищаем текст у **ePass2**: **ePass2.Text:= '';**
23. Теперь подумаем вот о чем. Окно редактора пользователей мы будем вызывать из главной формы, там же мы будем сохранять изменения, если администратор нажмет кнопку **Подтвердить** в окне редактора. А как из главной формы узнать - хочет ли

администратор сохранить изменения, или нет? Введем глобальную переменную **izmen** в модуль редактора пользователей:

```
var
  fEditor: TfEditor;
  izmen: boolean;
```

24. Теперь для формы редактора пользователей сгенерируем событие **OnShow**, в котором сразу пропишем**: izmen:= False;**
25. В событии нажатия на кнопку **Подтвердить** вписываем код:

```
procedure TfEditor.bAcceptClick(Sender: TObject);
begin
izmen:= True;
  Close;
end;
```

26. А при нажатии на кнопку **Отменить** просто закрываем форму:

```
procedure TfEditor.bCancelClick(Sender: TObject);
begin
Close;
end;
```

27. Теперь, если администратор закроет эту форму иначе, чем нажатием на **Подтвердить**, мы ничего предпринимать не будем. Осталось сделать проверки на то, ввел ли администратор логин нового пользователя, совпадают ли пароли в **ePass** и **ePass2**? Для этого сгенерируем событие **OnClose** для формы редактора:

```
procedure TfEditor.FormClose(Sender: TObject; var Action: TCloseAction);
begin
//если изменения не нужно делать, просто выходим:
  if not izmen then Exit;
  //Иначе делаем проверку. Если нет имени пользователя:
  if eUser.Text = '' then begin
    ShowMessage('Введите имя пользователя!');
    Action := caNone; //запрещаем покидать форму
    eUser.SetFocus; //переводим фокус на имя пользователя
  end;
  //Если пароль не '1111111111111111111', делаем проверку
  //совпадает ли ePass и ePass2:
  if ePass.Text <> '1111111111111111111' then
    if ePass.Text <> ePass2.Text then begin
      ShowMessage('Введите правильный пароль!');
      Action := caNone; //запрещаем покидать форму
      //очистим ePass и ePass2:
      ePass.Text:= '';
      ePass2.Text:= '';
      ePass.SetFocus; //переводим фокус на имя пользователя
    end;
end;
```

28. С формой редактора закончили работу, вернемся на главную форму. Не забудем сразу же командой **File - Use Unit** подключить к ней модуль **Editor**.

29. Реализуем добавление нового пользователя. Сгенерируйте обработчик нажатия на кнопку **Добавить**. Код обработчика следующий:

```
procedure TfMain.bAddUserClick(Sender: TObject);
begin
//вначале очистим данные, которые могут быть
  //в редакторе пользователей:
  fEditor.eUser.Text:= '';
  fEditor.ePass.Text:= '';
  fEditor.ePass2.Text:= '';
  fEditor.eName.Text:= '';
  fEditor.eMiddle.Text:= '';
  fEditor.eLast.Text:= '';
  //теперь показываем форму:
  fEditor.ShowModal;
  //если изменений делать не нужно, просто выходим:
  if not Editor.izmen then Exit;
  //иначе готовим компонент IBSS к созданию нового пользователя.
  IBSS.Active:= True;
  try
    //теперь вводим данные нового пользователя:
    IBSS.UserName:= fEditor.eUser.Text;
    IBSS.Password:= fEditor.ePass.Text;
    IBSS.FirstName:= fEditor.eName.Text;
    IBSS.MiddleName:= fEditor.eMiddle.Text;
    IBSS.LastName:= fEditor.eLast.Text;
    //вызываем метод AddUser, который добавляет пользователя:
    IBSS.AddUser;
  finally
    IBSS.Active:= False; //закрываем компонент
  end; //try
  //перечитываем информацию о пользователях:
  ReloadUsers;
end;
```

30. Сохраните проект, скомпилируйте его и попробуйте добавить нового пользователя. Пользователь должен появиться в окне **ListBox**, кроме того, он должен быть виден и в утилите **IBConsole**, в разделе **Users** дерева серверов.
31. Редактирование выбранного пользователя вызывается кнопкой **Редактировать.** Сгенерируйте этот обработчик. Вот его код:

```
procedure TfMain.bModifyUserClick(Sender: TObject);
var i: Integer; //счетчик
begin
  //если не один пользователь не выбран, просто выходим:
  if LB1.ItemIndex = -1 then begin
    ShowMessage('Выберите пользователя!');
    Exit;
  end
  //если выбран SYSDBA, тоже выходим:
  else if LB1.Items[LB1.ItemIndex] = 'SYSDBA' then begin
    ShowMessage('Редактировать пользователя SYSDBA нельзя.');
    Exit;
  end; //else if
  //иначе редактируем
  //установим у IBSecurityService выбранного пользователя:
  IBSS.Active:= True;
  IBSS.UserName:= LB1.Items[LB1.ItemIndex];
  //найдем в IBSS индекс нужного пользователя:
  for i:= 0 to IBSS.UserInfoCount - 1 do
    if IBSS.UserInfo[i].UserName =
       LB1.Items[LB1.ItemIndex] then break;
  //теперь i содержит индекс пользователя
  //заполняем редактор пользователей данными:
  fEditor.eUser.Text:= IBSS.UserInfo[i].UserName;;
  //раз пользователь уже есть, выводим единички в качестве пароля:
  fEditor.ePass.Text:= '1111111111111111111';
  fEditor.ePass2.Text:= '1111111111111111111';
  fEditor.eName.Text:= IBSS.UserInfo[i].FirstName;
  fEditor.eMiddle.Text:= IBSS.UserInfo[i].MiddleName;
  fEditor.eLast.Text:= IBSS.UserInfo[i].LastName;

  //теперь покажем редактор:
  fEditor.ShowModal;
  //если изменений делать не нужно, просто выходим:
  if not Editor.izmen then Exit;
  //иначе сохраняем их:
  try
    //Имя пользователя менять не будем.
    //Сохраним пароль, если там не единички:
    if fEditor.ePass.Text <> '1111111111111111111' then
      IBSS.Password:= fEditor.ePass.Text;
    //Далее сохраняем остальные параметры:
    IBSS.FirstName:= fEditor.eName.Text;
    IBSS.MiddleName:= fEditor.eMiddle.Text;
    IBSS.LastName:= fEditor.eLast.Text;
    //сохраняем изменения физически методом ModifyUser:
    IBSS.ModifyUser;
  finally
    IBSS.Active:= False;
  end; //try
  //перечитываем информацию о пользователях:
  ReloadUsers;
end;
```

32. Сохраните проект, скомпилируйте его и попробуйте отредактировать какого либо существующего пользователя, например, добавив к нему дополнительные данные

(фамилию, имя, отчество). После сохранения результата, в утилите **IBConsole** должны отобразиться эти данные.

33. Удаление пользователя - это самая простая операция с **IBSecurityService**. Ее код:

```
procedure TfMain.bDeleteUserClick(Sender: TObject);
var s: String; //для формирования строки
begin
  //если не один пользователь не выбран, просто выходим:
  if LB1.ItemIndex = -1 then begin
    ShowMessage('Выберите пользователя!');
    Exit;
  end;
  //иначе продолжаем. формируем запрос:
  s:= 'Вы действительно желаете удалить пользователя ' +
    LB1.Items[LB1.ItemIndex] + '?';
  //попросим подтверждения. выходим, если не подтвердили:
  if Application.MessageBox(PChar(s), 'Удаление пользователя',
    MB_YESNOCANCEL+MB_ICONQUESTION) <> IDYES then Exit;
  //если не вышли, значит удаляем пользователя
  IBSS.Active:= True;
  try
    //делаем пользователя текущим:
    IBSS.UserName:= LB1.Items[LB1.ItemIndex];
    //удаляем его методом DeleteUser:
    IBSS.DeleteUser;
  finally
    IBSS.Active:= False;
  end; //try
  //перечитываем информацию о пользователях:
  ReloadUsers;
end;
```

34. Сохраните проект, скомпилируйте его и попробуйте удалить какого либо существующего пользователя.

Внеаудиторная самостоятельная работа:

Составить письменно в тетради опорный конспект по основным командам программного администрирования баз данных в **InterBase**.